بـنـيـة بـيـانــات

يمكن وصف الحاسوب بأنه عبارة عن آلة تقوم بمعالجة البيانات. كما أن دراسة الحاسوب تتضمن دراسة كيفية تنظيم هذه المعلومات داخل الحاسوب، وكيفية معالجتها و كيفية الاستفادة منها. حيث يعتبر البت Bit الوحدة الأساسية للمعلومات و تستخدم الأرقام الثنائية 0 و1 لتمثيل Bit معين، و عدد ال Bits المستخدمة لتمثيل الرمز Character في حاسوب معين يسمى طول البايت Byte size. و بالتالي فان ذاكرة الحاسوب عبارة عن مجموعة من Bits، حيث يتم وضع كل مجموعة من Bits في ذاكرة الحاسوب في وحدات كبيرة تسمى Byte وفى بعض الحواسيب كل عدد من البايتات تجمع مع بعض و تسمى كلمة Word وكل وحدة من هذه الوحدات (بايت أو الكلمة) يكون لها عنوان وهو عبارة عن رقم و يطلق عليه موقع البايت في الذاكرة (location).

نوع البيانات Data Types:

إن كلمة data تعنى كل شي يمكن معالجته بواسطة برنامج حاسوبى .إذاً النوع البياني هو نوع من البيانات بالإضافة إلى قواعد كيفية معالجة هذه البيانات. حيث يتم تحديد النوع البياني بتحديد القيم من ذلك النوع و بتحديد العمليات المسموح بها على هذه القيم.

إن تعريف أي نوع في باسكال يحدد الآتي:

1. حجم الذاكرة التي سوف يحجزها المتغير المعرف من هذا النوع.
2. كيفية تمثيل البيانات داخل الحاسوب.

المصفوفات Arrays :

هي عبارة عن مجموعة منتهية و مرتبه من العناصر المتجانسة . نعنى بقولنا منتهية أي لها عدد محدود من العناصر .و مرتبة أي أن عناصرها مرتبة الأول فالثاني فالثالث ...وهكذا ، ومتجانسة أي أن جميع عناصرها من نفس النوع Type مثلا Char.
لا يمكن الجمع بين نوعين مختلفين في نفس المصفوفة.

الإعلان عن مصفوفة Array Declaration:

كل نوع من أنواع المصفوفات المستخدمة في البرنامج يجب أن يعلن عنها في قسم تعريف النوع بعد الكلمة المحجوزة Type وتكون الصيغة العامة Syntax لها في لغة باسكال كالآتي :

```
Type
     Name=Array[index type] of component type;
```

حيث أن:

- Name اسم تعريفي Identifier جديد لهذا النوع من المصفوفات.
- Index type نوع المؤشر الدليلى ، ويمكن أن يكون مدى صحيح Integer أو مدى حرف Character أو مدى منطقيBoolean أو مجموعة جزئية.
- Component type نوع المكونات :يمكن أن يكون أي نوع حقيقي أو صحيح أو... الخ.

هنالك عمليتان أساسيتان لمعالجة المصفوفات هي:

- تخزين البيانات.

- استرجاع البيانات.

بعد هذا الإعلان يمكن الإعلان عن العديد من المصفوفات من هذا النوع في قسم الإعلان عن المتغيرات.

أمثلة:

```
Var
  A:array[1..100] of integer;
```

- تم الإعلان عن المتغير A وهو عبارة عن مصفوفة مكونه من 100 من النوع الصحيح.

```
Const
      Num=100;
 Var
    A:array[1..num] of integer;
```

- تم الإعلان عن المتغير باستخدام الثابت Num في مدى المصفوفة.

```
Const
      Num=100;
Type
     Indextype=1..num;
 Var
    A:array[indextype] of integer;
```

- تم الإعلان عن مصفوفة باستخدام المدى من النوع Indextype الذي يستخدم الثابت Num

مما سبق يمكن أن نعرف حجم المصفوفة بأنه عبارة عن عدد المتغيرات الدليلية لها.

أمثلة لإعلانات مقبولة لبعض المصفوفات.

```
Const
      Min=0;
      Max=10;
Type
      Week=array[min..max]of Real;
      Degree=array[1..100] of integer;
```

ملاحظات :

1. يمكن أن تكون المصفوفة من النوع الحقيقي، و لكن المؤشر الدليلى لها لا يمكن أن يكون من النوع الحقيقي.
2. يمكن دمج الإعلان عن النوع و المتغير على سبيل المثال:

```
Var
   a:array[1..10] of real;
```

3. إن حجم المصفوفة فى لغة باسكال لا يتغير أثناء تنفيذ البرنامج.
4. لا يمكن قراءة كل المصفوفة كمتغير عادى و انما بإستخدام جملة for.

<u>استخدام عناصر المصفوفة</u>

المتغيرات الدليلية يمكن استخدامها في لغة باسكال كأي من المتغيرات الأخرى ،وسوف نرى فيما يلي من الأمثلة كيف أن حلقة For تسهل كثيراً من استخدام المصفوفات.

- قراءة مصفوفة وطباعتها:

```
Var
     Degree:array[1..10] of integer;
     I:integer;
Begin
     For I:=1 to 10 do
          Read(degree[I]);
     For I:=1 to 10 do
          Writeln(degree[I]);
End.
```

- إذا كان كل من A,B متغيراً يمثل مصفوفة من نفس النوع فإنه يمكن كتابة عبارة الإسناد التالية:

```
A:=B;
```

وتعنى إسناد كل عنصر من عناصر المصفوفة A إلى العنصر المقابل فى المصفوفة B

<u>المصفوفات ثنائية الأبعاد Two Dimensional Arrays:</u>

وهى مشابهة للمصفوفات الأحادية البعد في طريقة الإعلان عنها،وطباعتها
مثلاً:

```
Type
    Matrix=arrary[1..5,1..10]of integer;
Var
    A:matrix;
```

الإعلان أعلاه يفيد أن A أسم مصفوفة مكونة من خمسة صفوف و عشرة أعمدة وبالتالي فإن عدد عناصرها يساوي 5x10 ، وأن كل عناصرها صحيحة، وتسمى المتغيرات الدليلية،وهى تمثل الاتى:

A[1,1]……a[1,10],A[2..1]..A[5 ,1]…A[5,10]

لقراءة هذه المصفوفة يمكن أن نستخدم دورة For كالتالي:

```
For I:=1 to 5 do
  For j:= 1 to 10 do
       Read(a[I,j]);
```

وطباعتها تتم بطريقة مشابهة لطريقة القراءة أي تتم طباعة الصف الأول فالثاني وهكذا.

<u>مثـــال:</u>

اكتب برنامج يقرأ عناصر المصفوفتين A و B المكونتين من 3 صفوف و4 أعمدة ثم يجمع عناصر المصفوفة الأولى مع الثانية ويحتفظ بحاصل الجمع في المصفوفة C التي أبعادها أيضاً 3 X 4 ، ثم يطبع عناصر المصفوفة C

```
uses crt؛
type
   My_Array=array[1..3,1..4] of integer؛
var
   A,B,C:My_Array؛
   i,j:integer؛
begin
    clrscr؛
                        {reading the first Array}
    for i:=1 to 3 do
       for j:=1 to 4 do
       begin
       write('Enter A[',i,',',j,'] : ');
       readln(A[i,j]);
       end؛
                        {reading the second Array}
    for i:=1 to 3 do
       for j:=1 to 4 do
       begin
        write('Enter B[',i,',',j,'] : ');
       readln(B[i,j])؛
       end؛
                        {calculating the Third Array}
    for i:=1 to 3 do
       for j:=1 to 4 do
       begin
           C[i,j]:=A[i,j]+B[i,j]؛
       end؛
                        {Printing The Third Array}
       for i:=1 to 3 do
       begin
           for j:=1 to 4 do
               write(C[i,j]:10)؛
           writeln؛
       end؛
    readln
end.
```

Records   الــسجلات

السجلات تشبه المصفوفات في كونها مجموعة من البيانات المتعلقة ببعضها البعض، وتختلف عنها في إمكانية احتوائها علي بيانات مختلفة في النوع ،فمثلاً يمكننا استخدام سجل لتخزين معلومات متباينة النوع عن شخص معين كاسمه و عمره وتاريخ الميلاد...
وبدلاً من استخدام المؤشر الدليلى – كما هو الحال في المصفوفات – فإننا سوف نستخدم ما يعرف بالحقل Field.

الإعلان عن السجلات Record Declaration:

في البداية نعلن عن هيكلية السجل، وذلك من خلال الإعلان عن نوع السجل بعد الكلمة المحجوزة Type بعد ذلك نعلن عن متغير لسجل واحد أو أكثر من ذلك النوع ، و يكون ذلك بعد الكلمة المحجوزة Var .

مثال :

افترض أننا نريد تخزين المعلومات التالية عن الطلاب (أسمه ،الفصل الدراسي ،الدفعة)

```
Name = Array[1..10] of char;
Class=integer;
Year=integer;
```

المثال التالي يتم الإعلان عن سجل من النوع Student يحوى على ثلاثة حقول مختلفة

```
Const
    Max=5;
Type
     Str= array[1..max] of char;
     Student= record
          Name :str;
          Class:integer;
          Year:integer;
     End;
Var
    Stud:Student;
```

استخدام حقول السجلات Usage of field:

بعد الإعلان عن السجلات ، فإنها يمكن أن تستخدم كأي متغير، فيمكن أن تسند إليها قيم من نفس النوع،أو أن تكون ضمن أوامر القراءة أو الكتابة .وسوف نستعرض هنا ثلاث طرق للتعامل مع السجلات وهى:

1. **باستخدام النقطة:**

للتعامل مع حقل لسجل ما نكتب اسم السجل ونتبعه بنقطة، ثم نكتب اسم الحقل المراد التعامل معه، وبذلك يتم التعامل مع الحقل كأي متغير آخر.

مثلاً:

```
Stud.name:='ali  ';
Stud.class:=4;
Stud.year:=2001;
```

2. **استخدام بنية With-do:**

الطريقه السابقة قد تكون مملة فى بعض الأحيان، و خصوصاً عندما يكون اسم السجل كبيراً ويحتوى على العديد من الحقول .إن بنية **With - do** تتيح لنا أن نهمل اسم السجل .
الصيغة العامة:

With var **do** statement

حيث ان :

Var	اسم متغير لسجل
Statement	جملة او مجموعة من الجمل و خلال تلك الجمل فإن أي حقل يتم تحديد سجله بعد كلمة With لا داعى أن يسبق باسم سجله.

فبناء على ما سبق من إعلانات يمكن أن نكتب:

```
With stud do
 begin
    name:='ahmed';
    birth.day:=22;
    birth.mon:=1;
    birth.year:=1975;
    place:='sudan';
end;
```

3. الاسناد الكلى للسجل:

يمكن أن يسند سجل إلى سجل آخر بعبارة إسناد واحدة، ولكن بشرط أن يكون لهما نفس نوع السجل، وكمثال على ذلك إذا كان لدينا الإعلان التالي:

```
Var
     Stud1,stud2:student;
```

يمكننا ان نكتب عبارة الاسناد التالية:

```
Stud1:=stud2;
```

مثال

اكتب برنامج يقرأ بيانات سجل الطالب المكون من الرقم والاسم ثم يطبع تلك البيانات

```
uses crt;
Type
  Student = record
    No:integer;
    Name: string;
  end;
var
  Stud: Student;
begin
    clrscr;
    write('Enter Student No: ');
    readln(Stud.No);
    write('Enter Student Name: ');
    readln(Stud.Name);

    with Stud do
    begin
        write('Student : ',No,'  ',Name);
    end;

Readln
end.
```

مثال

اجعل البرنامج أعلاه يستقبل بيانات 5 طلاب بدلاً من طالب واحد

```
uses crt;
Type
  Student = record
    No:integer;
    Name: string;
  end;
var
  Stud: array[1..5] of Student;
  i:integer;
begin
    clrscr;
    for i:= 1 to 5 do
    begin
```

```
        writeln('Enter No of student ',i);
        readln(Stud[i].No);
        writeln('Enter Name of Student ',i);
        readln(Stud[i].Name);
    end;
    for i:=1 to 5 do
        writeln('Student ',i,' : ',Stud[i].No,'  ',Stud[i].Name);

Readln
end.
```

بـنـيـة الـبـيـانـات

هي طرق لبناء البيانات.حيث يتم تنظيم البيانات في شكل عناصر مركبة وإتاحة عمليات لمعالجة هذه العناصر.

النوع البياني Data type يتم الإعلان عنه في جزء التصريحات في البرنامج.

بينما بنية البيانات توجد فقط في ذهن المبرمج واحتمال أن تظهر في جزء التصريحات في البرنامج وربما لا

أشكال البيانات المجردة Abstract Data Types:

هي نوع من أنواع البيانات تكون في شكل كبسولة تعرف بداخلها بنية بيانات معينة أمثال المصفوفات أو السجلات أو المؤشرات.تربط هذه البنية بمجموعة من المعالجات التي تغير و تعالج قيم عناصر هذه البنية .

إن لإشكال البيانات المجردة كثير من التطبيقات في مجالات علوم الحاسوب المختلفة، فالمكدسات تستخدم في نظم التشغيل المختلفة و الصفوف تستخدم في عمليات المحاكاة كما أن الأشجار تدخل بصورة واضحة في بناء المترجمات. يمكننا اعتبار الأشكال المجردة بأنها البداية الحقيقية لطريقة البرمجة الموجهة للكائنات Object Oriented Programming language OOPL.

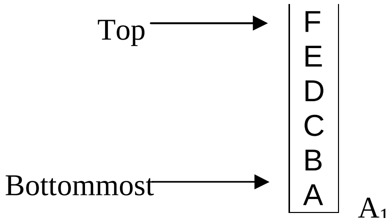
المكدسات Stacks

تعريف المكدسة Define Stack:

المكدسة هي قائمة من البيانات تحفظ بطريقة خطية، يتم فيها حذف و إضافة البيانات من طرف واحد يسمي القمة Top.

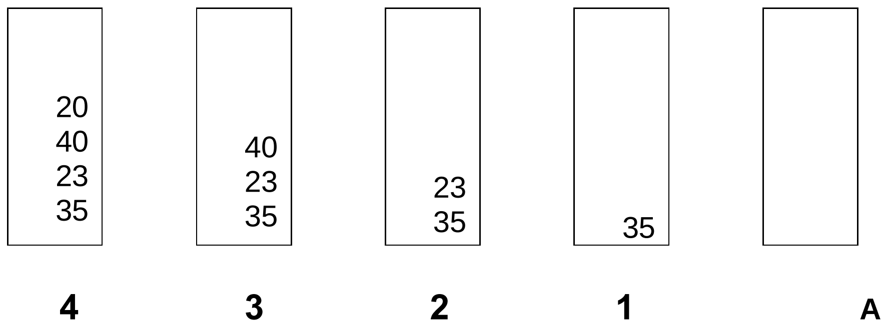
ليكن لدينا المكدسة S=(a1,..,an) فنقول أن العنصر A1 في القاع Bottommost ، وأن العنصر A_{i-1} أعلى من العنصر A_i حيث 1<I<n ونقول أن a_n في قمة المكدسة.الشكل التالي يوضح مفهوم المكدسة.

A_n



من تعريف المكدسة فإنه إذا أضفنا العناصرA,B,C,D,E,F إلي المكدسة بنفس هذا الترتيب،فإن العنصر الذي يكون بإمكاننا حذفه هو العنصر F أي أن آخر عنصر يتم إدخاله في المكدسة هو الذي يحذف أولاً .و لهذا السبب يطلق علي هذا النوع من أشكال البيانات المجردة أسم (LIFO) اختصاراً ل Last IN First Out وبالمثل في حالة الإضافة فإن العنصر الجديد يصبح في القمة.

مثلاً افترض أن لدينا المكدسة A و التي تسع خمسة أرقام و أردنا أن نجري عملية أضافة للقيم 35 و 23 و 40 و 20 فإن الأضافة تكون في اتجاه واحد وهو أعلى المكدسة



المكدسة فارغة

و لأجراء عملية السحب من المكدسة ، تسحب العناصر بترتيب عكسي لترتيب دخولها المكدسة أي يتم سحب الرقم 20 أولاً ثم 40 ثم 23 ثم 35 .

عمليات المكدسة Operations on Stacks:

من المثال السابق نلاحظ أن هناك عمليتان رئيسيتان تعمل كلاً منهما علي تغيير حالة المكدسة ، وهما عملية الإضافة Push وعملية الحذف Pop ، فإذا افترضنا أن مكدسة اسمها S ونريد أن نضيف لها عنصر a ،فإننا نرمز لعملية الإضافة Push(S,a)، وبعد إجراء هذه العملية فإن العنصر a يصبح في قمة المكدسة، ولما كانت عملية الحذف لا تتم إلا من قمة المكدسة فقط فإنه يتم حذف العنصر الذي يقع في قمة المكدسة.

لنفرض الآن أننا قمنا بإجراء عملية الحذفpop و كررناها حتى آخر عنصر في المكدسة، فإننا نحصل على مكدسة خالية Emptystack لذلك قبل إجراء عملية الحذف يجب التأكد من أن المكدسة ليست خالية وذلك بواسطة العملية Empty(s) والتي تخبرنا ما إذا كانت المكدسة خالية أم لا، فإذا كانت المكدسة خالية فإن Empty(s) تعطى القيمة المنطقية True وإلا فإنها تعطى القيمة المنطقية False.

لنفرض الآن أن لدينا مكدسة خالية ،وقمنا بإجراء عملية حذف فإننا نحصل علي Underflow وبالتالي فإنه يجب فحص المكدسة بواسطة العملية Empty(s) قبل إجراء عملية الحذف .و بالمثل إذا كان لدينا مكدسة ممتلئة و قمنا بإجراء عملية إضافة Push فإننا نحصل على OverFlow.

و لذلك يمكن استخدام عملية Full(s) قبل عملية الإضافة Push لتفادي Overflow، وهي ترجع القيمة المنطقية true إذا كانت المكدسة ممتلئة، كما ترجع القيمة المنطقية False إذا كانت غير ذلك.

بناء المكدسات باستخدام المصفوفات في لغة باسكال : Stacks implementation Using Arrays in Pascal:

المكدسات هي مجموعة من العناصر،وفي لغة باسكال هناك المصفوفات، وهي عبارة عن مجموعة من البيانات، لذلك لتمثيل المكدسات في لغة باسكال فإننا نعرف نوع من المتغيرات باسم Stack بواسطة مصفوفة.

ولكننا نلاحظ أن هناك اختلافاً كبيراً في طبيعة كل من المكدسة والمصفوفة، فعدد عناصر المصفوفة ثابت، ويحدد منذ الإعلان عن المصفوفة، ولا يمكن للمستخدم أن يغير هذا العدد في الغالب، أما بالنسبة للمكدسة فإن حجمها يتغير باستمرار كلما حدثت عملية حذف أو إضافة.

لذا فإننا سوف نعتبر المصفوفة كبيت للمكدسة، و نعلن عن مصفوفة ذات مدى يستوعب أكبر حجم نتوقع أن تصل إليه المكدسة ، وأثناء تنفيذ الحذف والإضافة فإن المكدسة سوف تتمدد وتنكمش في إطار هذا المدى المحدد سلفاً نحدد طرفاً من أطراف المصفوفة كقاع ثابت للمكدسة، بينما سوف تتحرك للقمة صعوداً أو هبوطاً كلما حدث حذف أو إضافة للمكدسة ، وهذا يستلزم منا تعريف متغير آخر ليحدد الموقع الحالي لقمة المكدسة.

<u>الإعلان عن المكدسة:</u>

مما سبق نجد أن المكدسة يمكن الإعلان عنها في لغة باسكال كسجل يحتوى على حقلين مصفوفة لاحتواء عناصر المكدسة، ومتغير صحيح لتحديد الموقع الحالي لقمة المكدسة في المدى المحدد للمصفوفة كما هو موضح أدناه

```
Const
  Maxstack=100;
Type
   Stack=record
      Item:array[1..maxstack] of integer;
      Top:0..maxstack;
   End;
Var
  S:stack;
```

وكما هو واضح فإن القمة top يجب أن تكون من النوع Integer بين 0 و Maxstack لأن قيمتها تحدد موقع قمة المكدسة،فمثلاً إذا كانت Top=3 فهذا يعنى أن المكدسة تحتوى على ثلاثة عناصر item[1],item[2],item[3] وعندما تحدث عملية حذف Pop من المكدسة، فإن القمة TOP تتغير إلى 2، أما إذا حدثت عملية إضافة push فإن القمة سوف تزداد الى 4، وitem[4] سوف تأخذ قيمة العنصر الجديد الذي تمت إضافة.

لقد علمنا من قبل أن المكدسة الخالية لا تحتوى علي أية عنصر، وقد نستطيع الاستدلال علي ذلك عندما تكون القمة top مساوية للصفر،ولجعل المكدسة في حالتها الابتدائية(مكدسة خالية)، فإننا نجعل القمة Top مساوي للصفر. وبالمثل فعندما تكون المكدسة ممتلئة نستطيع الاستدلال علي ذلك عندما تكون القمة top مساوية للثابت maxstack.

الدالة Empty:

لاختبار ما إذا كانت المكدسة خالية أم لا،فإننا نستخدم الدالةempty و التي ترجع القيمة true إذا كانت المكدسة خالية ،و القيمة False إذا كانت غير ذلك

```
Function empty(s:stack):boolean;
Begin
        If  s.top=0 then
              Empty:=true
         Else
              Empty:=false;
End;
```

ويتم الاختبار في البرنامج الرئيسي كالآتي:

```
If empty(s) then
    Writeln('the stack is empty')
 Else
     Writeln('the stack is not empty');
```

الدالة Full:

وهي لاختبار ما إذا كانت المكدسة ممتلئة أم لا فترجع القيمة true إذا كانت المكدسة ممتلئة والقيمة false إذا كانت غير ذلك.

```
Funciton full(s:stack):Boolean;
Begin
   If  s.top=maxstack then
       Full:=true
   Else
        Full:=fal se;
End;
```

و يتم الاختبار في البرنامج الرئيسي كالاتي:

```
If full(s) then
    Writeln('the stack is full')
 Else
     Writeln('the stack is not empty');
```

<u>خوارزمية الحذف من المكدسة:</u>

1. اختبار ما إذا كانت المكدسة خالية أم لا،فإذا كانت غير خالية استمر في تنفيذ بقية الخطوات،وإلا فارجع إلى البرنامج المنادى.
2. احذف العنصر الذي يقع في قمة المكدسة .
3. اطرح واحد من قيمة قمة المكدسة (Top).
4. ارجع العنصر المحذوف إلي البرنامج المنادى.

```
Function Pop(var s:stack):integer;
Begin
    If empty(s) then
        Writeln('stack underflow')
    Else
      Begin
         Pop:=s.item[s.top];
         s.top:=s.top-1;
      End;
End;
```

و لمناداة هذه الدالة في البرنامج الرئيسي نكتب

```
X:=empty(s);
```

عملية الإضافة للمكدسة:

1. اختبر إذا كانت المكدسة ممتلئة أم لا،إذا لم تكن ممتلئة استمر في تنفيذ الخطوات وإلا فارجع للبرنامج المنادى.
2. أضف واحد إلي قمة المكدسة .
3. أسند العنصر المراد إضافته إلي قمة المكدسة.

```
 Pocedure push(var  s:stack; x:integer);
Begin
    If  full(s) then
        Writeln('stack overflow')
    Else
      Begin
            s.top:=s.top+1;
            s.item[s.top]:=x;
      End;
End;
```

وعليه يصبح برنامج المكدسة كاملاً كالتالي

```
Const
  Maxstack=100;
  Type
     Stack=record
        Item:array[1..maxstack] of integer;
        Top:0..maxstack;
```

```
      End;
Var
  S:stack;
Function empty(s:stack):boolean;
  Begin
    If  s.top=0 then
      Empty:=true
    Else
      Empty:=false;
  End;
Function full(s:stack):Boolean;
  Begin
     If  s.top=maxstack then
        Full:=true
     Else
        Full:=false;
  End;
Function Pop(var s:stack):integer;
  Begin
     If empty(s) then
       Writeln('stack underflow')
     Else
        Begin
          Pop:=s.item[s.top];
          s.top:=s.top-1;
        End;
  End;
Procedure push(var  s:stack; x:integer);
   Begin
     If  full(s) then
        Writeln('stack overflow')
     Else
        Begin
          s.top:=s.top+1;
          s.item[s.top]:=x;
        End;
   End;

begin
end.
```

تطبيقات على المكدسات

1- استقبال جملة وطباعتها معكوسة

يتم ذلك باستقبال الحروف حتى ضغط المستخدم لمفتاح Enter ، ومفتاح الإدخال أيضاً أحد رموز لوحة المفاتيح ويمكن استقباله على أنه حرف، ورقمه بحسب شفرة آسكي هو 13 لذلك نختبر ما إذا كان المدخل هو الحرف رقم 13 لاستمرار الحلقة بالطريقة التالية

```
    while ch<>chr(13) do
    begin
        push(S,ch);
        read(ch);
    end;
```

ويصبح البرنامج كاملاً كالآتي

```
uses crt;
Const
    Maxstack=100;
    Type
        Stack=record
            Item:array[1..maxstack] of char;
            Top:0..maxstack;
        End;
Var
   S:stack;
   ch: char;

Function empty(s:stack):boolean;
   Begin
      If  s.top=0 then
         Empty:=true
      Else
         Empty:=false;
   End;
Function full(s:stack):Boolean;
   Begin
        If  s.top=maxstack then
           Full:=true
        Else
           Full:=false;
   End;
Function Pop(var s:stack):char;
   Begin
       If empty(s) then
```

```
         Writeln('stack underflow')
       Else
          Begin
            Pop:=s.item[s.top];
            s.top:=s.top-1;
          End;
  End;
Procedure push(var  s:stack; c:char);
   Begin
      If  full(s) then
         Writeln('stack overflow')
      Else
          Begin
            s.top:=s.top+1;
            s.item[s.top]:=c;
          End;
   End;

begin
    clrscr;
    read(ch);
    while ch<>chr(13) do
    begin
        push(S,ch);
        read(ch);
    end;

    while not empty(S) do
    begin
        ch:=pop(s);
        write(ch);
    end;


Readln
end.
```

2- نقل عناصر مكدسة إلى مكدسة أخرى دون الإخلال بالترتيب

S	S1
12	
34	
54	
23	
32	
70	
44	
15	
60	

نلاحظ عند أخذ أول عنصر في قمة المكدسة S وهو 12 وإضافة هذا العنصر إلى المكدسة الثانية S1، سوف يصبح العنصر 34في قمة المكدسة S وعندما نستمر على هذا المنوال نلاحظ أن كل عناصر المكدسة S انتقلت إلى المكدسة الثانية S1 ولكن بصورة معكوسة عليه فان عملية النقل لم تتم بصورة صحيحة .

(سوف تكون بصورة **S** نقل إليها عناصر المكدسة **temp** إذن لابد من وجود مكدسة مؤقتة معكوسة كما هو مبين أدناه)

S	Temp	S1
	60	
	15	
	44	
	70	
	32	
	23	
	54	
	34	
	12	

وستكون معكوسة مع عناصر S1 إلى المكدسة temp ونقوم بنقل عناصر المكدسة المؤقتة . وتصبح عندئذ المكدسة منقولة من S ومرتبة مع عناصر المكدسة tempالمكدسة المؤقتة وهو يوضح المطلوب S1 إلى المكدسة Sالمكدسة

S	Temp	S1
		12
		34
		54
		23
		32
		70
		44
		15
		60

وتتم أولاً قراءة المصفوفة الأولى S عن طريق الحلقة التالية

```
for i:=1 to 10 do
begin
    write('Enter Number into stack : ');
    readln(no);
    push(S,No);
end;
```

ثانياً نقل عناصر المصفوفة S إلى المصفوفة Temp عن طريق الحلقة التالية

```
while not empty(S) do
begin
    No:=pop(s);
    push(Temp,No);
end;
```

ثالثاً نقل عناصر المصفوفة Temp إلى المصفوفة S1

```
while not empty(Temp) do
begin
    No:=pop(Temp);
    push(S1,No);
end;
```

وأخيراً عرض عناصر المصفوفة S1 عن طريق الحلقة

```
while not empty(S1) do
    write(pop(S1):7);
```

ويصبح البرنامج كاملاً

```
uses crt;
Const
    Maxstack=100;
    Type
        Stack=record
            Item:array[1..maxstack] of integer;
            Top:0..maxstack;
        End;
Var
   S,S1,Temp:stack;
   i,No: integer;

Function empty(s:stack):boolean;
   Begin
      If  s.top=0 then
         Empty:=true
      Else
         Empty:=false;
   End;
Function full(s:stack):Boolean;
   Begin
        If  s.top=maxstack then
            Full:=true
        Else
            Full:=false;
   End;
Function Pop(var s:stack):integer;
   Begin
       If empty(s) then
          Writeln('stack underflow')
       Else
           Begin
              Pop:=s.item[s.top];
              s.top:=s.top-1;
           End;
   End;
Procedure push(var  s:stack; x:integer);
    Begin
        If  full(s) then
           Writeln('stack overflow')
        Else
```

```
          Begin
            s.top:=s.top+1;
            s.item[s.top]:=x;
          End;
   End;

begin
    clrscr;
    for i:=1 to 10 do
    begin
        write('Enter Number into stack : ');
        readln(no);
        push(S,No);
    end;
    while not empty(S) do
    begin
        No:=pop(s);
        push(Temp,No);
    end;

    while not empty(Temp) do
    begin
        No:=pop(Temp);
        push(S1,No);
    end;

    while not empty(S1) do
        write(pop(S1):7);

Readln
end.
```

3- فحص وزن الأقواس داخل التعبير الرياضي

وفحصنا هذا معني فقط بالأقواس المفتوحة والمغلقة داخل التعبير الرياضي ومنها نصل إلى أن وضع الأقواس موزون إذا كان كل قوس مفتوح ")" يقابله قوس مغلق "(" . وللتحقق من وزن التعبير نستخدم الخوارزمية التالية:

1- صمم مكدسة للحروف
2- نفترض في البداية أن التعبير موزون Balanced عن طريق وضع متغير بولياني بنفس الاسم يحمل القيمة True
3- ما دام التعبير balanced ولم تنته السلسلة الحرفية بعد إذن قم بالآتي
   أ- اقرأ الحرف التالي
   ب- إذا كان الحرف قوس مفتوح أدخله بالمكدسة

ج- إذا كان قوس مغلق قم بالآتي
a – أخرج أول قيمة من المكدسة
b- إذا كانت المكدسة خالية ضع balanced=false
4- بعد نهاية السلسلة الحرفية قم بفحص المكدسة، إن كان بها عناصر ضع balanced=false

```
    Balanced:=True;
    read(ch);
    while(Balanced) and (ch<>chr(13)) do
    begin
        if ch='(' then push(S,ch)
        else if ch=')' then
        begin
           if empty(s) then Balanced:=False
           else ch:=pop(s);
        end;
        read(ch);
    end;
    if not empty(s) then Balanced:=False;
    if Balanced then writeln('Balanced Exepresion')
    else writeln('Non Balanced Exepresion');
```

بينما يصبح البرنامج كاملاً بالشكل التالي:

```
uses crt;
Const
   Maxstack=100;
   Type
       Stack=record
          Item:array[1..maxstack] of char;
          Top:0..maxstack;
       End;
Var
  S:stack;
  ch: char;
  Balanced:boolean;
Function empty(s:stack):boolean;
  Begin
    If  s.top=0 then
       Empty:=true
    Else
       Empty:=false;
  End;
Function full(s:stack):Boolean;
```

```
  Begin
      If  s.top=maxstack then
          Full:=true
      Else
          Full:=false;
  End;
Function Pop(var s:stack):char;
  Begin
     If empty(s) then
        Writeln('stack underflow')
     Else
         Begin
           Pop:=s.item[s.top];
           s.top:=s.top-1;
         End;
  End;
Procedure push(var  s:stack; c:char);
   Begin
      If  full(s) then
         Writeln('stack overflow')
      Else
          Begin
            s.top:=s.top+1;
            s.item[s.top]:=c;
          End;
   End;

begin
     clrscr;
     Balanced:=True;
     read(ch);
     while(Balanced) and (ch<>chr(13)) do
     begin
          if ch='(' then push(S,ch)
          else if ch=')' then
          begin
             if empty(s) then Balanced:=False
             else ch:=pop(s);
          end;
          read(ch);
     end;
     if not empty(s) then Balanced:=False;
     if Balanced then writeln('Balanced Exepresion')
```

```
    else writeln('Non Balanced Exepresion');

Readln
end.
```

Infix And Postfix -3

عندما نريد جمع المتغيرين A , B فإننا نعتبر التعبير الرياضي التالي A+B وهذا يعني تطبيق العملية "+" على المعاملين A و B وهذا التمثيل يسمى Infix ولكن هنالك طريقتين أخريين لتمثيل ذلك التعبير هما:

+AB prefix
AB+ postfix

ويستخدم الحاسوب هاتين الطريقتين في تفسير معاني العمليات الرياضية حتى يستطيع الفصل بين الأولويات لأن تفسير التعابير الرياضية يعتمد على فهم أي العمليات ينبغي إنجازها أولاً، كمثال فإن عمليات الضرب والقسمة تسبق عمليات الطرح والجمع حتى إن سبقتها الأخيرتان في الترتيب وللفهم اعتبر التعبير (A+B*C) يجب هنا تنفيذ عملية الضرب قبل الجمع.
وإذا أردنا تحويل التعبير أعلاه إلى Postfix فإننا نفصل عملية الضرب أولاً في قوس A+(B*C) ثم نضع علامة الضرب أمامهما للإشارة إلى أنها تنجز أولاً (*A+(BC ثم نضع عملية الجمع بعد عملية الضرب لمراعاة الأولويات +(*A(BC وعند هذه المرحلة نكتشف أنه لم يكن هنالك داع للأقواس لأن التعبير أصبح جاهزاً بصيغة Postfix ويراعي الأولويات ABC*+
والقاعدة الوحيدة للتحويل من Infix إلى Postfix هي أن العمليات ذات الأولوية الأعلى تطبق قبل العمليات ذات الأولوية الأقل، وهنا يجب أن ننتبه إلى أن التعابير بداخل الأقواس يجب أن تتم أولاً قبل أن نربطها بالعمليات خارج الأقواس

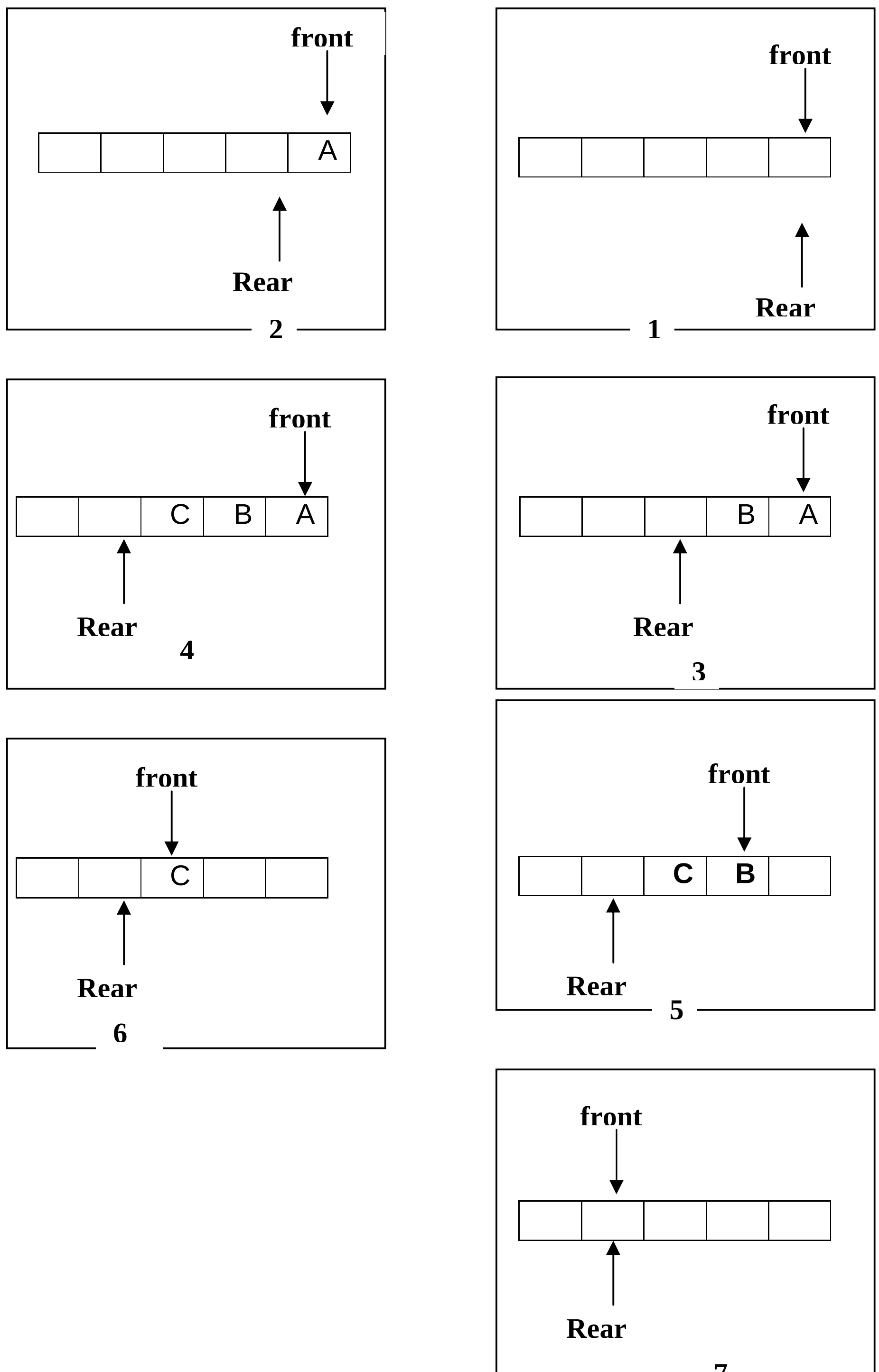
Infix	Postfix
A+B	AB+
A+B-C	AB+C-
A+B/C	ABC/+
A*(B+C)	ABC+*
(A+B) / (C-D)	AB+CD-/
A/B*C-D+E/F*(G+H)	AB/C*D-EF/GH+*+
A-B/(C+D)	ABCD+/-

الـصـفـوف Queues

تعريف الصف:

هو قائمة من العناصر تحفظ بطريقة خطية، خطية ولا يختلف عن المكدسات إلا في العنصر الأول في الإدخال هو العنصر الأول في الإخراج (الحذف)أيضاً وهذا الأسلوب يعرف بأسلوب (FIFO) First In First Out في الوصول إلى البيانات، وهذا يعنى أن حذف العناصر يتم من طرف واحد ويسمى المقدمة أو الرأس Front ، وإضافة العناصر أيضا تتم من الطرف الآخر الذي يسمى الذيل أو المؤخرة Rear .

لتوضيح فكرة الصف دعنا نتأمل الأشكال التالية:



في الحالة 1 نجد نه لدينا صف خالي تتساوى فيه قيمة الرأس و المؤخرة ، و في الشكل 2 تمت إضافة العنصر A في مكان المؤخرة Rear و زيادة قيمة Rear ليؤشر إلى المنطقة الخالية التي تلي منطقة آخر عنصر تمت إضافته ، و في الشكل 4 تمت إضافة عنصر آخر إلى الصف في مكان Rear الجديد و زيادة قيمته ليؤشر إلى المكان الخالي الذي يلي مكان العنصر الذي تمت إضافته ، وفي الشكل 5 نجد أن هناك عنصر قد تم حذفه من الصف، ولما كان حذف العناصر من الصف يتم من المقدمة،لذلك تم حذف العنصر A،وتحرك المؤشر Front خطوة إلي الأمام و بذلك يصبح العنصر b في المقدمة ، و في الحالة 6 تم حذف عنصر آخر من الصف و تقدم المؤشر Front خطوة أخرى لتصبح C في المقدمة ، و كذلك في الحالة 7 نجد أن عنصراً آخر تم حذفه و تقدم المؤشر Front خطوة أخرى ليصبح مع Rear في مكان واحد و يصبح الصف خالي من جديد.

ولهذا السبب يسمى هذا الشكل من أشكال البيانات المجردة (FIFO) اختصاراً First In First Out.

عمليات الصفوف Queue Operations:

ليكن لدينا الصف q فإن هناك أربع عمليات يكمن إجراؤها عليه ،وهى الإضافة Insert(q,x)،وفيها يتم إضافة العنصر X إلى مؤخرة الصف Q ، وعملية الحذف X:=remove(q); وفيها يتم حذف العنصر الذي يقع في مقدمة الصف Q ووضعه في X وعملية Empty(q) وهى تقوم بفحص الصف، وترجع القيمة True إذا كان الصف خالياً، و False إذا كان غير ذلك ، وأخيراً العملية Full(q) وهي تقوم بفحص الصف، وترجع القيمة True إذا كان الصف ممتلئاً ، وFalse إذا كان غير ذلك.

إذا لم يكن هنالك حد لحجم الصف ، فإن عملية الإضافة يمكن إجراؤها من غير قيود، أما عملية الحذف فيتم إجراؤها إذا لم يكن الصف خالياً فقط.و إذا تم إجراؤها علي صفٍ خالٍ، فإن النتيجة تكون **Underflow** ومن الواضح أن عملية Empty يمكن إجراؤها في أي لحظة، و كذلك عملية Full.

الإعلان عن الصف:

سوف نستخدم المصفوفة لتمثيل الصف في لغة باسكال ، و ذلك بعد إضافة بعض الضوابط التي تحكم عملية الإضافة و الحذف من الصف ، و سوف نستخدم متغيرين front و Rear لتحديد العنصر الأول و الأخير في الصف ، بحيث تشير front إلى موقع العنصر الأول و Rear إلى الموقع الذي يقع خلف العنصر الأخير مباشرةً في الصف ، أي أن موقع Rear سيكون خالياً دوماً ، و بالتالي فإن صف من الأعداد الصحيحة يمكن تمثيله كالتالي:-

```
Const;
MaxQueue = 5;
Type
Queue=Record
item : array[1..MaxQueue]of integer;
Front,Rear:1..MaxQueue;
End;
```

و تذكر أن المصفوفة محدودة العناصر مما يتسبب في امتلاء الصف و حدوث OverFlow .

الدالة Empty :

المتغير Rear سوف يكون خالياً دوماً و هو يمثل مؤخرة الصف ، أي أن الصف يكون خالياً إذا كان Front = Rear و بذلك يمكن تمثيل العملية empty كالتالي:

```
function Empty(q:Queue):Boolean;
begin
if q.Front =q.Rear then Empty:=True
else Empty:=False;
end;
```

الدالة Full :

يكون الصف ممتلئاً إذا كان مؤشر المؤخرة Rear يشير إلى الموقع الذي يلي آخر عنصر يمكن أن تحتويه المصفوفة ، أي حد المصفوفة + 1 .

```
function Full(q:Queue):Boolean;
begin
if q.Rear=MaxQueue+1 then Full:=True
else Full:=False;
end;
```

عملية الحذف من الصف :

عند حذف عنصر من الصف فيجب أولاً التأكد من أن الصف ليس خالياً و بعد ذلك يتم حذف العنصر الموجود في موقع المقدمة Front ثم تتم زيادة قيمة مؤشر المقدمة بواحد و يمكن تمثيل عملية الحذف كالتالي:-

```
Function Remove(var q:Queue):integer;
begin
if Empty(q) then writeln('Queue UnderFlow');
else
begin
Remove:=q.item[q.Front];
q.Front:=q.Front+1;
end;
end;
```

عملية الإضافة:-

عند إضافة عنصر يجب أولاً التأكد من أن الصف ليس ممتلئاً ثم بعد ذلك تتم إضافة ذلك العنصر في الموقع الذي يؤشر إليه مؤشر المؤخرة Rear و تتم زيادة قيمة ذلك المؤشر بمقدار واحد ليؤشر إلى مكان خالي

و يمكن تمثيلها كالتالي:-

```
Procedure Insert(var q:Queue;X:integer);
begin
if Full(q) then writeln('Queue OverFlow')
else
begin
q.item[q.Rear]:=X;
q.Rear:=q.Rear+1;
end;
end;
```

و لاختبار برنامج الصف يمكن تطبيق هذا البرنامج الذي يستقبل 5 أرقام ويطبعهم

```
uses crt;
Const
   MaxQueue = 5;
Type
   Queue=Record
        item : array[1..MaxQueue]of integer;
        Front,Rear:1..MaxQueue;
   End;

function Empty(q:Queue):Boolean;
begin
    if q.Front =q.Rear then Empty:=True
else Empty:=False;
end;


function Full(q:Queue):Boolean;
begin
    if q.Rear=MaxQueue+1 then Full:=True
    else Full:=False;
end;

Function Remove(var q:Queue):integer;
begin
    if Empty(q) then writeln('Queue UnderFlow')
else
          begin
          Remove:=q.item[q.Front];
          q.Front:=q.Front+1;
      end;
end;
```

```
Procedure Insert(var q:Queue;X:integer);
begin
    if Full(q) then writeln('Queue OverFlow')
    else
      begin
          q.item[q.Rear]:=X;
          q.Rear:=q.Rear+1;
      end;
end;

var
  q:Queue;
  No,i:integer;

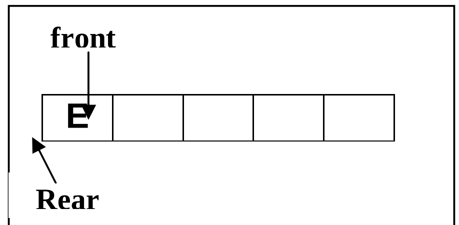
begin
     clrscr;
     q.rear:=1;
     q.front=1
     writeln('Enter 5 Numbers');
     for i:=1 to 5 do
     begin
         write('Enter No ',i,' : ');
         readln(No);
         Insert(q,No);
     end;

     writeln('The Numbers you Entered is :');

     while not empty(q) do
     begin
         writeln(Remove(q));
     end;
     readln
end.
```

الصف الدائري Circular Queue

في تطبيقنا للصف بالطريقة السابقة ظهرت مشكلة صغيرة و هي الامتلاء الكاذب للصف ، خذ مثلا الرسم أدناه و هو يوضح حالة أحد الصفوف حيث تم حذف جميع العناصر ما عدا العنصر الأخير ، و لكن بالرغم من وجود أماكن كثيرة فارغة إلا أننا لا نستطيع إضافة أي عنصر ، لأن المؤشر Rear موجود في نهاية المصفوفة محققاً لشرط الامتلاء



و لحل هذه المشكلة تم ابتكار ما يسمى بالصف الدائري و هو عملية السماح بتخزين العناصر في المصفوفة عند وجود أي مكان فارغ . و تتم هذه العملية باعتبار أن الصف عبارة عن دائرة، أي عند وصول مؤشر المؤخرة إلي نهاية الصف يتم تحويله إلى بدايته ليتم الإدخال من البداية و نفس الحال بالنسبة لمؤشر المقدمة

ولعمل ذلك الصف الدائري سنحتاج إلى تعديل دالة الامتلاء لتصبح بالشكل التالي

```
if (q.Rear+1) mod (MaxQueue+1)= q.front then Full:=True
```

و دالة الإضافة ستستطيع إضافة العناصر حتى إذا وصل مؤشر المؤخرة إلى نهاية الصف وذلك بإجراء التعديل التالي

```
q.Rear:= (q.Rear+1) mod (MaxQueue+1);
```

و بالمقابل لا بد من تعديل دالة الحذف أيضاً لتصبح بالشكل التالي:-

```
q.Front:=(q.Front+1) mod (MaxQueue+1);
```

ويمكن كتابة نص البرنامج كاملاً كما يلي:-

```
uses crt;
const
   MaxQueue = 5;
Type
```

```
   Queue=Record
        item:array[1..MaxQueue]of integer;
        Front,Rear:1..MaxQueue;
   end;

function Empty(q:Queue):boolean;
begin
    if q.Front =q.Rear then Empty:=True
    else Empty:=False;
end;

function Full(q:Queue):boolean;
begin
    if (q.Rear+1) mod (MaxQueue+1)= q.front then Full:=True
    else Full:=False;
end;

Function Remove(var q:Queue):integer;
begin
    if Empty(q) then writeln('Queue UnderFlow')
    else
      begin
          Remove:=q.item[q.Front];
          q.Front:=(q.Front+1) mod (MaxQueue+1);
      end;
end;

Procedure Insert(var q:Queue;X:integer);
begin
    if Full(q) then writeln('Queue OverFlow')
    else
      begin
          q.item[q.Rear]:=X;
          q.Rear:= (q.Rear+1) mod (MaxQueue+1);
      end;
end;



var
  q:Queue;
  No,i:integer;
```

```
begin
     clrscr;
           q.Rear:=1;
           q.Front:=1;
     writeln('Enter 10 Numbers');
     for i:=1 to 5 do
     begin
         write('Enter No ',i,' : ');
         readln(No);
         Insert(q,No);
     end;
     Insert(q,11);

     writeln('The Numbers you Entered is :');

     for i:= 1 to 6 do
     begin
         writeln(Remove(q));
     end;

     writeln('Enter 10 Numbers');
     for i:=1 to 5 do
     begin
         write('Enter No ',i,' : ');
         readln(No);
         Insert(q,No);
     end;
     Insert(q,11);

     writeln('The Numbers you Entered is :');

     for i:= 1 to 6 do
     begin

         writeln(Remove(q));
     end;
     readln
end.
```

المؤشرات Pointers

نوع مؤشّر pointer يحدّد متغيرا يحوي عنوان متغير آخر في الذاكرة ذو نوع بيانات محدد (أو نوع غير محدد). لذا فإن متغير المؤشر و بطريقة غير مباشرة يشير إلى قيمة.

تعريف نوع مؤشّر لا يستلزم كلمة مفتاحية معينـة، ولكنـه يستعمل حرفا خاصـا بدلا من ذلك .
الحرف أو الرمز الخاص هو علامة الادراج :(^) caret

type
 PointerToInt = ^Integer;

حالما عرّفت متغيرا مؤشرا، يمكنك أن تخصص له العنوان الخاص بمتغير آخر من نفس النوع، باستخدام العامل :@

```
uses crt;
var
p:^integer;
x:integer;
begin
  clrscr;
    x:=20;
    p:=@x;
    write(p^);
  readln
end.
```

عندما يكون لديك المؤشّر *P* ، فإنه بواسطة التعبير *P* أنت تشير إلى موقع في الذاكرة يشير إليه المؤشر، و بواسطة التعبير P^ أنت تشير إلى المحتويات الفعلية في موقع الذاكرة هذا .لهذا السبب في التوليف السابق فإن *P^* تطابق *X* أي تعني قيمة ما يؤشر إليه المتغير P

الكود P=@X يعني أن المؤشر p يؤشر إلى المتغير x أو بعبارة أخرى يحمل عنوان المتغير x في ذاكرة الحاسوب

بدلا من الإشارة إلى موقع ذاكرة موجود، يمكن للمؤشر أن يشير إلى مساحة جديدة في الذاكرة يتم تخصيصها بصورة حيّة (في منطقة من محيط الذاكرة) بواسطة الإجرائية *New.* في هذه الحالة، و عند انتهاء حاجتك للمؤشر، عليك أيضا أن تتخلص من الذاكرة التي قمت بحجزها، باستدعاء الوظيفة . *Dispose*

```
uses crt;
var
p:^integer;
begin
```

```
  clrscr;
    new(p);
    p^:=20;
    write(p^);
    dispose(p);
  readln
end.
```

إذا كان المؤشر خاليا من أية قيمة، يمكنك تخصيص قيمة *nil* لا شيء له، بعدها تستطيع اختبار خلوّ المؤشر لمعرفة إذا ما هو حاليا يشير إلى قيمة. و هذا يستعمل عادة، لأن التأشير الخاطئ بمؤشر فارغ يسبّب انتهاكا لحرمة الدخول (access violation).

يمكن أيضاً الوصول إلى المتغيرات بدلالة مؤشراتها فإذا كان p يؤشر إلى x بالعبارة p:=@x تصبح قيمة x هي قيمة ما يؤشر إليه p بالطريقة p^:=x وللتوضيح خذ المتغيرين m و n الذين بدءا بالقيمتين 15 و 25 ثم غيرنا قيمهما بدلالة المؤشر لتصبحا 12 في المثال التالي

```
uses crt;
var
p:^integer;
m,n:integer;
begin
  clrscr;
    m:=15;
    n:=20;
    p:=@m;
    p^:=12;
    p:=@n;
    p^:=12;
    write('m =',m,' and
n=',n);
  readln
end.
```

الخلاصة أن العلامة ^ تجعل من المؤشر متغيراً

ويمكن أن نضع مؤشر لكافة أنواع البيانات ولنأخذ المثال التالي للإشارة إلى سجل الموظفين

```
Type
  Pemployee = ^Temployee;
  Temployee = record
                 Name     : string[10];
                 Position : char;
```

```
                 Salary    : longint;
               end;
var
  p : pemployee;
begin
  new(p);
  p^.name:='Yassir';
  p^.position:='S';
  p^.salary:=3000;
  writeln(p^.name,' ',p^.position,' ',p^.salary);
  dispose(p);
end.
```

<u>مدخل إلي المؤشرات في هياكل البيانات:</u>

سنتعرض هنا إلي كيفية استعمال لغة باسكال في إنشاء **بنية بيانات متحركة Dynamic Data Type.** ونقصد بقولنا متحركة أي أنها تنمو Grow أثناء تنفيذ البرنامج .

و بنية البيانات المتحركة عبارة عن مجموعة من العناصر تسمى عقداً Nodes و هي في الحقيقة سجلات Record مرتبطة ببعضها البعض بواسطة المؤشرات.

و بخلاف المصفوفات التي تشكل مخزناً لعدد محدود من العناصر، فإن بنية البيانات المتحركة يمكن أن تتمدد و تنكمش خلال تنفيذ البرنامج، ويتوقف ذلك على حاجة البرنامج لتخزين البيانات، و بمعنى آخر فإنه عند الإعلان عن مصفوفة يتم حجز Allocate مساحة محددة من الذاكرة لا نستطيع تجاوزها عند استخدام المصفوفة ، و عند تجاوز هذه المساحة يحدث ما يسمى بمشكلة Array Overflow .وهذه الحالة لا تحدث عند استخدام المؤشرات لأن المساحة المتاحة تكون مفتوحة.

إن بينة البيانات المتحركة تستخدم لتخزين بيانات دائمة التغيير في عالم حقيقي مثال لذلك قائمة المسافرين ، كما أنها توصف بالمرونة ، فمن السهل جداً إضافة بيانات جديدة عن طريق إنشاء عقدة جديدة ، وإدخالها بين عقدتين موجودتين أصلاً .

<u>عبارة New والمتغير من نوع مؤشرPointer:</u>

في المصفوفات، كنا نقوم بحجز مكان في الذاكرة لعناصر المصفوفة عن طريق الإعلان عن مصفوفة ذات حجم معين، وهذا بالطبع يرغمنا على تحديد العدد الأقصى لعناصر المصفوفة في حالة بنية البيانات المتحركة، فإننا لا نعلم كم سيكون حجم تلك البنية،أي عدد العقد التي سوف تحتويها، لذلك يجب علينا حجز مكان للعقدة في الذاكرة عند حاجتنا إليها. وبطريقة ما نربط تلك العقدة بباقي البنية ،إن عبارة New تستعمل لحجز مكان لعقدة جديدة.

كيف يمكننا إنشاء و استعمال عقدة جديدة؟ في المصفوفات كان هناك المؤشر الدليلى يدلنا على مكان تلك العقدة في حالتنا هذه نجد أن لغة باسكال توفر نوعاً خاصاً من المتغيرات يسمى مؤشر Pointer.

مثال:

```
Type

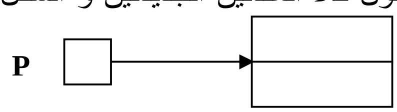
   Nodepointer=^node;
   Node=record
          Name:string;
          Age:integer;
   End;
 Var
      P,q,r:nodePointer;
```

هذه الإعلانات تعلن عن سجل من النوع Node و عن مؤشر من النوع Nodepointer يؤشر إلى السجلات من النوع Node. المتغيرات R,Q,P معرفة على أنها مؤشرات من النوع NodePointer

إن الجملتين

```
New(p);
New(Q);
```

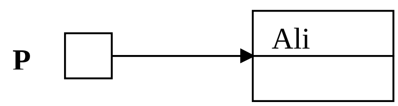
تحجزان أماكن لسجلين مشارُ إليها بالمؤشرات P,Q و حيث أن P,Q من النوع NodePointer فإن هذه السجلات الجديدة- المشار إليها ب Q,P– يجب أن تكون من النوع Node كما هو موضح أدناه، إن قيمة المتغير مؤشر Pointer variable هي في الحقيقة عنوان Address في الذاكرة لعقدة محددة.سوف نمثل المؤشر برسم سهم باتجاه العقدة التي يشير إليها المؤشر .لاحظ أنه إلي الآن لم يتم تعريف حقول كلا العقدتين الجديدتين و الشكل التالي يجعلنا نتخيل ما شرحناه آنفاً



لتعريف الحقل Name في العقدتين المشار إليهما بالمؤشرين P,Q فإنه يمكن استخدام الجملتين:

```
P^.name:='ali';
Q^.name:='ahemd';
```

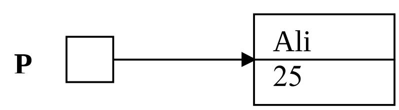
لتعريف حقل name لـP^(أي العقدة المشار إليها بالمؤشرP) وحقل Name لـQ^ (أي العقدة المشار إليها بالمؤشر Q) ليصبح لدينا الشكل التالي:



الشكل يبين أن P يشير إلى السجل من النوع Node، والذي يحتوى حقله الأول على السلسلة String 'Ahemd' و لتعريف الحقل Age في العقدتين السابقتين فإنه يمكن استخدام الجملتين:

```
P^.age:=24;
Q^.age:=19;
```

كما هو موضح في الشكل التالي



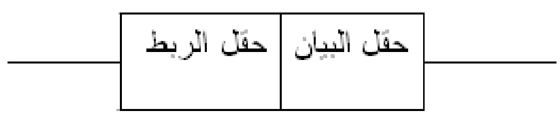
من المهم فهم الاختلاف في استعمال P و P^ في البرنامج.P متغير من النوع مؤشر Type nodePointer، ويستخدم لتخزين عنوان بنية بيانية من النوع Node في الذاكرة، ويمكن أن تأخذ قيمة جديدة من خلال عبارة إسناد مؤشر أو تنفيذ عبارة New .أما P^ فهي اسم السجل المشار إليه بواسطة P، ويمكن أن يعامل كأي سجل آخر في لغة باسكال .أما الحقولP^.name و P^.age فهي تستخدم لتخزين البيانات في ذلك السجل.

القوائم المتصلة

القوائم المتصلة هي سلسلة من العقد بحيث كل عقدة ترتبط بعقدة لاحقة لها، ماعدا العقدة الأخيرة التي ترتبط بالقيمة صفر **(Nil)** والتي تشير إلي نهاية القائمة. هنا نحتاج إلي توضيح بعض المصطلحات التي تستخدم في القوائم المتصلة وهي:

العقدة Node

وهي عبارة عن سجل أو كائن في القائمة يمثل وحدات مختلفة من الذاكرة المعنونة ويحتوي على حقول/حقل للبيانات وحقل آخر يحمل عنواناً يؤشر للعقدة التالية في القائمة، وليست بالضرورة أن تكون العقد متلاصقة فيزيائياً في الذاكرة والشكل الآتي يوضح ذلك:



العقدة أعلاه تحتوي على حقلين:

حقل بيان

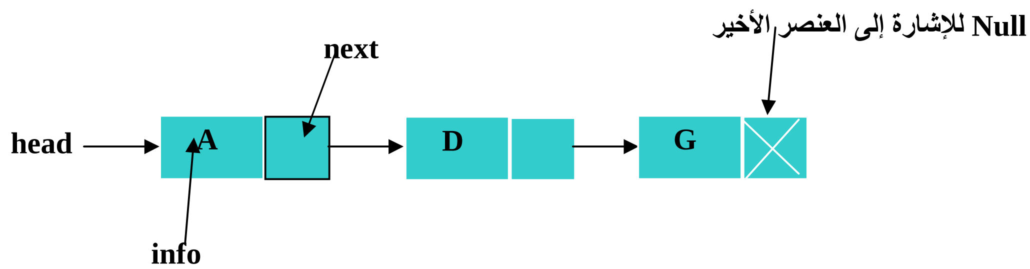
هو الحقل الذي يحمل العنصر الحقيقي في القائمة وبيانات هذا العنصر قد تكون غير متجانسة مثل : رقم موظف ، اسم طالب أو سجلاً كاملاً.

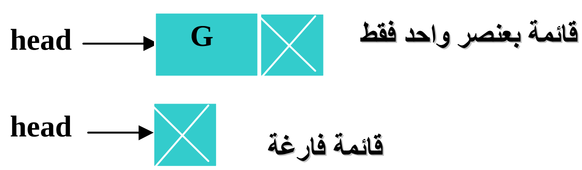
حقل ربط

يحمل عنوان العقدة التالية في القائمة ويعرف باسم المؤشر (**Pointer**) .

المؤشر Pointer

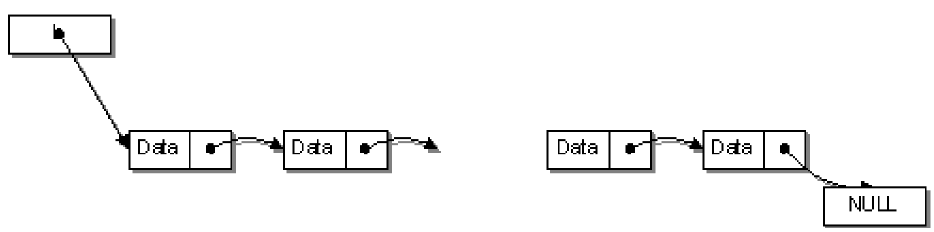
يحتوي المؤشر علي قيمة بيانية معينة تحدد موقع تخزين العنصر التالي في القائمة وغالبا ما تكون مكتوبة بالنظام الثنائي أو السادس عشري.





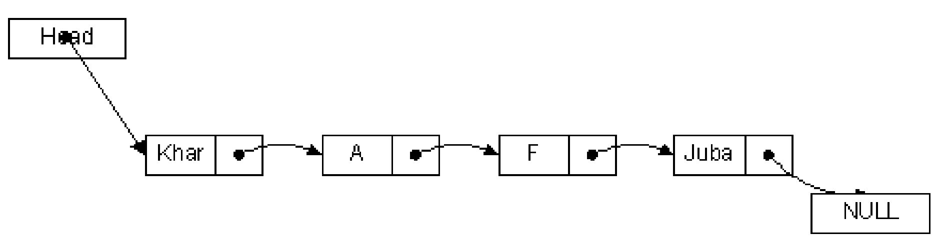
القائمة أحادية الاتجاه One Way linked List

هي قائمة تشير كل عقدة فيها إلي العقدة التي تليها وبالتالي تكون العناصر مرتبطة مع بعضها البعض على شكل سلاسل ويوجد مؤشر لأول عقدة في القائمة والعقدة الأخيرة تؤشر إلي (لا شيء) .



مثال: إذا كان لدى شركة الخطوط الجوية السودانية خط رحلات يبدأ من مدينة **Khar** وينتهي بمدينة **Juba** لاتجاه واحد بدون عودة مروراً بالمدينة **A** والمدينة **F** .

ويمكن تمثيل ذلك في شكل قائمة متصلة أحادية الاتجاه كما هو مبين أدناه:



الوصول إلي عنصر يتم التحرك له من القيمة الابتدائية **Head** وجعلها تؤشر إلى العقدة الأولى في القائمة ثم الانتقال إلي العقدة التالية التي يؤشر إليها المؤشر وهكذا إلى أن نصل إلي العقدة المحددة، مثلاً إذا أردنا الوصول إلى المدينة **F** يجب أن نتحرك من مدينة **Khar** مروراً بالمدينة **A** حتى نصل إلى المدينة **F** .

إنشاء القائمة المتصلة

هنالك عدد من الدوال يمكن أن تتعامل مع القوائم المتصلة كعمليات، نذكر من هذه الدوال الأتي:

<1> الإنشاء :

<2>. الإضافة **Insertion** :

- ✓ إدراج عقدة جديدة في نهاية القائمة .
- ✓ إدراج عقدة جديدة في بداية القائمة .
- ✓ إدراج عقدة جديدة بين عقدتين .

<3>. البحث .

<4>. الحذف :

- ✓ حذف العقدة الأولى .
- ✓ حذف العقدة الأخيرة .
- ✓ حذف عقدة اختيارية .

<5>. إيجاد الطول .

<6>. عرض عناصر القائمة .

<7>. نسخ قائمة .

الإعلان عن القائمة المتصلة:

يتم إنشاء القائمة المتصلة عن طريق الشفرة التالية

```
type

List_Type=integer;

List_Pointer=^ListNode;

ListNode=record

          Data:List_Type;

          Link:List_Pointer;

        end;

var

  Head:List_Pointer;
```

حيث يتم تعريف نوع بيانات نسميه List_Type يحمل نوع بيانات العناصر داخل القائمة المتصلة ، وهنا وضعناه نوع الأعداد الصحيحة Integer والغرض من ذلك هو إمكانية تعديل نوع البيانات في القائمة المتصلة بتعديل واحد بدلاً من تعديل كل دوال وإجراءات القائمة المتصلة. ثم عرفنا مؤشراً يؤشر إلى سجل القائمة ListNode ، ثم عرفنا السجل الذي يمثل كل عقدة من عقد القائمة المتصلة وهو يحتوي على العنصر Data الذي يحمل قيمة العقدة ، ثم عرفنا المؤشر Link الذي يشير إلى العقدة التي تلي العقدة الحالية. وفي قسم المتغيرات عرفنا مؤشراً يشير إلى رأس القائمة Head وهو عنوان بداية القائمة المتصلة بإشارته إلى العقدة الأولى بالقائمة.

وفيما يلي سنصمم برنامجاً تجريبياً لإدخال عناصر في قائمة متصلة يستقبل عدد من العناصر ويدخلها بالقائمة المتصلة وينتهي إدخال العناصر بإدخال الرقم 0

```
type
List_Type=integer;
List_Pointer=^ListNode;
ListNode=record
          Data:List_Type;
          Link:List_Pointer;
        end;
var
  Head:List_Pointer;
  No:integer;
procedure Fillrest(Last:List_Pointer);
```

```
begin
    writeln('Enter No');
    read(no);
    while no<>0 do
    begin
        Last:=Last^.link;
        Last^.data:=no;

        writeln('Enter No');
        read(no);
    end;
    Last^.link:=nil;
end;
begin

    writeln('Enter No');
    read(no);
    if no<>0 then
    begin
        head^.data:=No;
        fillrest(head);
    end
    else head:=nil;

end.
```

طباعة عناصر القائمة المتصلة

لطباعة العناصر يمكننا البدء من Head الذي يشير دوماً إلى أول عنصر بالقائمة وإرساله إلى الدالة PrintList التي ستستنسخ منه نسخة أخرى ثم تقوم بتحويل نسختها الخاصة من Head من عقدة إلى أخرى دون التأثير على Head الأصلي التابع للبرنامج الرئيسي.

ويتم التنقل عبر العقد باستخدام المؤشرات بالشفرة التالية Head:=Head^.Link

```
procedure printlist(head:listpointer);
begin
  while head<> nil do
   begin
     writeln(Head^.data);
     head:=head^.link;
   end;
end;
```

وفيما يلي برنامج كامل لاختبار الدوال والإجراءات الموضحة أعلاه

```
Type

  Listpointer=^listnode;

  Listnode=record

       Data:integer;

       Link:listpointer;

  End;

Var

  Head:listpointer;  {pointer to list head}

  ch,x:integer;



procedure printlist(head:listpointer);

begin

  while head<> nil do
```

```
    begin
       writeln(Head^.data);
       head:=head^.link;
    end;
end;


procedure creatlist(var head:listpointer);
var
     firstNo:integer;
procedure fillrest(last:listpointer);
      {append new nodes to the end of alist}
var
    nextNo:integer;
begin
  read(nextNo);
  while nextNo <> 0 do
    begin
       new(last^.link);
       last:=last^.link;
       last^.data:=nextNo;
       read(nextNo);
    end;
   last^.link:=nil;
```

```
end;
begin  {display the instruction to user}
   writeln('enter each Numbers on a line');
   writeln('enter 0 when done');
   read(firstNo);
   if firstNo=0 then
      head:=nil
   else
      begin  {build list}
           new(head);
           head^.data:=firstNo;
        fillrest(head);
      end;
end;


function search(head:listpointer;target:integer):listpointer;
var
  found:boolean;
begin
    found:=false;   {target not founds}
   while not found and (head<> nil) do
    if head^.data= target then
         found:=true
```

```
    else
      head:=head^.link;   {move down the list}
    if found then
      search:=head          {success}
    else
      search:=nil;        {failure}
end;

 Begin
      repeat
        writeln('1-Create list');
         writeln('2-Print list');
         writeln('3-Search list');
        writeln('4- Exit');
         writeln('Inter your choice');
        read(ch);
        writeln;
        case ch of
         1:begin
               creatlist(head);
               writeln;
             end;
         2:begin
```

```
                printlist(head);
                writeln;
            end;
         3:begin
              writeln('inter numebr to seachr...');
              readln(x);
              if search(head,x)=nil then
                writeln ('not found...')
              else
               writeln('found');
              writeln;
            end;
      end;
   until  ch=4;
end.
```

النداء الذاتي للدوال Recursion function

النداء الذاتي للدالة هو أن تعيد الدالة تكرار نفسها أو تستدعي الدالة نفسها مجدداً حيث يتم استدعاءها داخل الدالة بنفس اسم الدالة إلى أن يتحقق شرط التوقف للدالة المستدعاة وهنالك بعض الملاحظات التي يجب أن تذكر وهي:

1- تستدعى الدالة نفسها داخل الدالة بواسطة معاملات مختلفة في كل مرة يتم الاستدعاء.

2- في النداء الذاتي للدوال إذا لم يتم وضع شرط التوقف فستستدعي الدالة نفسها إلى ما لا نهاية function will call itself forever

3- عند اكتمال آخر استدعاء للدالة يقوم المترجم بإرجاع كل القيم التي نتجت عند الاستدعاء الذاتي وتبدأ بآخر قيمة نتجت عند الاستدعاء حتى أول قيمة.

استخدم النداء الذاتي للدوال في حل كثير من المسائل الرياضية بكثرة وذلك لكونها أنسب طريقة برمجياً للتعامل مع جمع وضرب المتسلسلات وترتيبها.

مثال (1)

اكتب برنامجاً لإيجاد مضروب أي عدد صحيح N باستخدام النداء الذاتي

```
function f(n:integer):integer;

begin

    if n<=1 then f:=1

    else f:=n*f(n-1);

end;
```

وإذا حاولنا تتبع مسار الدالة أعلاه بافتراض أن القيمة المرسلة لها كانت 4 مثلاً أي n=4 فإننا سنحصل على النداءات الموضحة أعلاه مع العلم بأن النداءات تتوقف فقط في حال توقف الشرط n<=1 والنداءات هي

```
F(4)= 4 * f(3)
        F(3)=3* f(2)
                F(2)=2*f(1)
                        F(1)=1
                F(2)=2*1
        F(3)=3*2*1
F(4)=4*3*2*1
```

مثال (2)

اكتب برنامجاً لحساب فابوناتشي Fibonacci number وذلك من خلال العلاقة التالية

F(0) = 0

F(1) = 1   f(2)=0+1=2

F(n) = F(n-1) + F(n-2)

صمم دالة تستقبل الحد من سلسلة فيبوناتشي ثم تقوم بإيجاد قيمة ذلك الحد

```
var
   a:integer;
function f(n:integer):integer;
begin
    if (n=1) or (n=2) then f:=n
    else
       f:=f(n-1)+f(n-2);
end;
begin
   read(a);
       write(f(a));
readln
end.
```

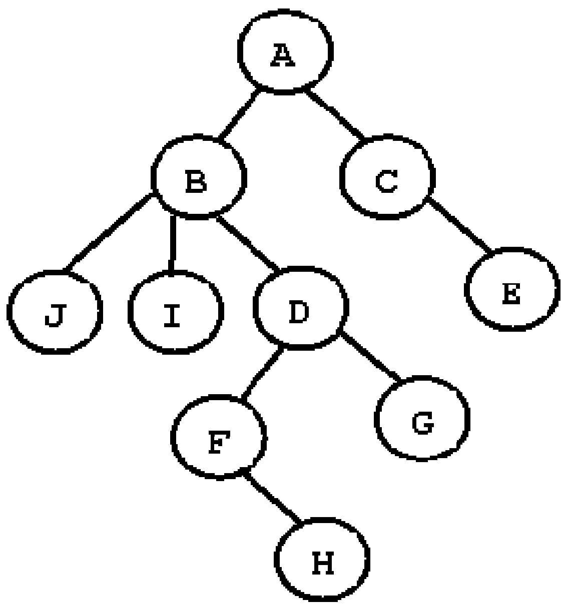
نلاحظ عند استدعاء الدالة في البرنامج الرئيس من خلال عبارة F (N) نرسل للدالة العدد N وتستقبله في المتغير n ثم بعد ذلك يتم اختبار الشرط هل هذا العدد يساوى 2 أو 1 فإذا كانت الإجابة بنعم فان الدالة سوف تعيد هذه القيمة إلى البرنامج الرئيس أما إذا كانت اكبر منهم فإنها سوف تستدعي الدالة مرة أخرى من خلال التعبير التالي:

F:= f (n-1)+f (n-2);

هنا الاستدعاء الذاتي للدالة قد تم أكثر من مرة وفى كل مرة بمعامل مختلف عن الأخرى

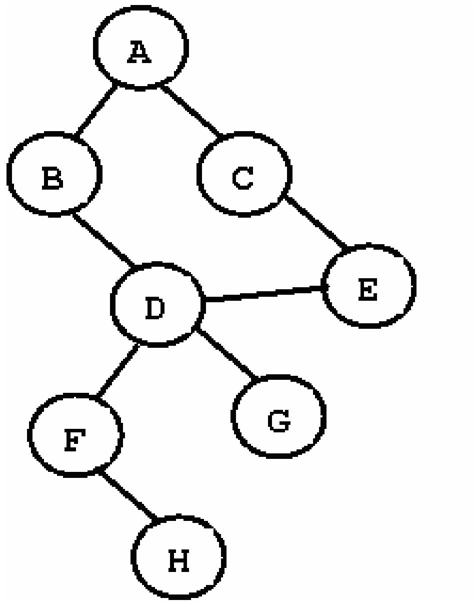
الأشجار Trees

تعتبر الأشجار واحدة من نماذج هياكل البيانات التي تنتمي لمجموعة أكثر شمولا تسمى المخطط **GRAPH** وتتكون الشجرة من مجموعة عقد **Nodes** تربط بينها ممرات **Edges** وهنالك عقدة وحيدة في المستوى الأول للشجرة نطلق عليها العقدة الرئيسية أو الجزر **Root** بحيث تكون الممرات منها للعقد التي تليها فقط وهذه الممرات لا تمثل دورة، والشكل رقم (1) يبين نموذجاً للشجرة



الشكل رقم (1) نموذج للشجرة

أما إذا احتوى الهيكل على دورة فإنه يصبح مخططاً وليس شجره والشكل رقم (2) يبين أن الهيكل احتوي علي الدورة (**A B D E C**) لهذا فإن الهيكل لا يمثل شجرة وإنما يسمى مخططاً أو بياناً.



الشكل رقم (2) مخطط احتوى على دورة

لا تقف هياكل الأشجار في أنها مفيدة فقط بل يمكن القول بأنها تعتبر أفضل من جميع هياكل البيانات الأخرى من حيث المزايا التي تعود من استخدامها فنجد سرعة تنفيذ العمليات الخاصة بالبحث والحذف و الإضافة عالية جداً مقارنة مع هياكل البيانات الأخرى.

تعريفات

الأبناء Children

هي العقد التي تنحدر مباشرة من العقدة العلوية وترتبط بها بواسطة مؤشرات الربط يطلق علي العقدة العلوية اسم الأب Parent والعقد إلي تنحدر منها أبناء Children وفي الشكل رقم (1) تسمى العقدة B والعقدة C أبناء الجذر A وتسمي العقدة F والعقدة G أبناء للعقدة D التي تسمي الأب Parent وإذا كان عدد الأبناء اثنين يسمى الابن من جهة اليمين بالابن الأيمن Right Child والابن من جهة اليسار بالابن الأيسر Left Child وفي الشكل رقم (1) تسمي العقدة G الابن الأيمن Right Child والعقدة F بالابن الأيسر Left Child للعقدة D .

الممر Edge

هو عبارة عن مؤشر الربط الذي يربط عقدتين مثل مؤشر الربط الذي يربط بين الجذر والابن.

الورقة Leaf

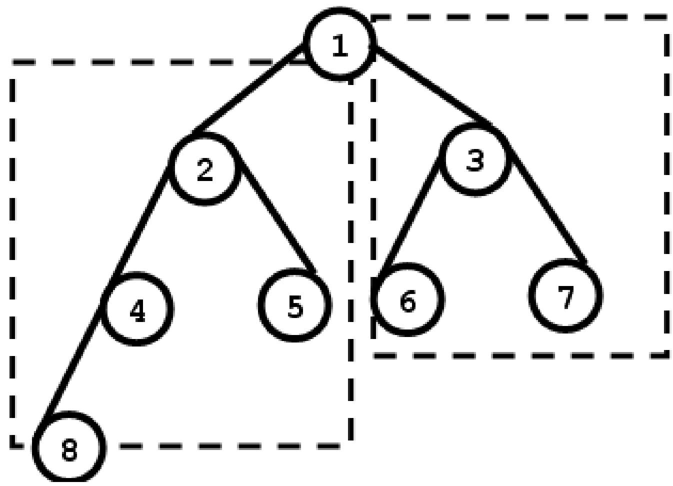
هي عبارة عن عقدة لا تحتوي على أبناء وفي الشكل رقم (1) تسمى العقد J و I و H و G و E أوراق.

المسار Path

هو عبارة عن مجموعة متتالية من الممرات وإذا وصل المسار إلى ورقة يسمى المسار غصناً **Branch** وفي الشكل رقم (1) يسمي **المسار A<-B<-d<-F<-H** غصن.

الشجيرة الفرعية Tree Sub

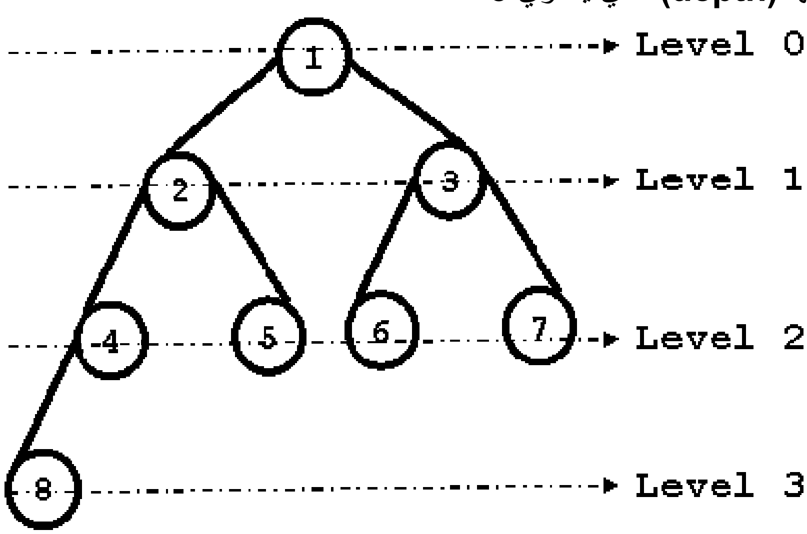
يطلق اسم الشجيرة الفرعية على أي جزء من الهيكل الشجري والشكل رقم (3) يوضح شجيرات فرعية منبثقة من الشجرة الأساسية.



الشكل رقم (3) الشجيرات الفرعية

المستوي Levels

يستخدم مستوى الشجرة أو عمقها للدلالة على أكبر عدد من العقد الموجودة في أغصان الشجرة أو أكبر مستوي لأي عقدة بالشجرة والشكل رقم (4) يبين المستويات المختلفة للشجرة وعمقها **(depth)** الذي يساوي **3**

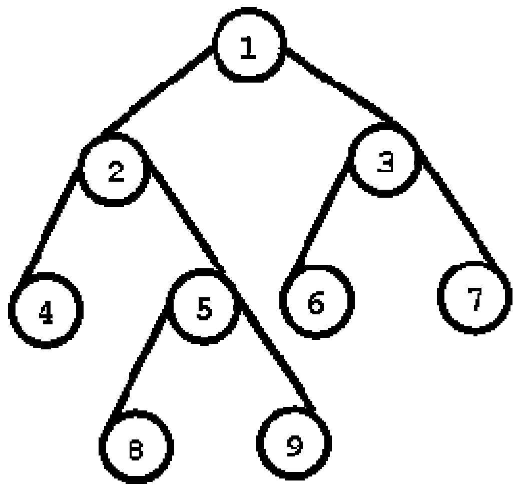


.

الشكل رقم (4) عمق ومستوى الشجرة

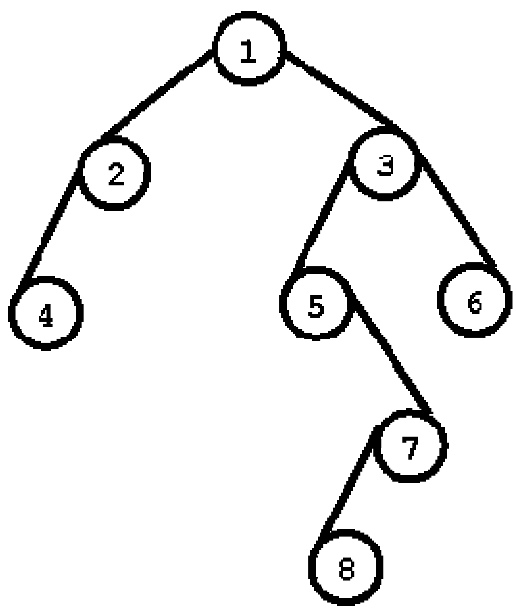
مثال (1)

احسب عمق الأشجار التالية



(أ)

العمق 3=(depth)

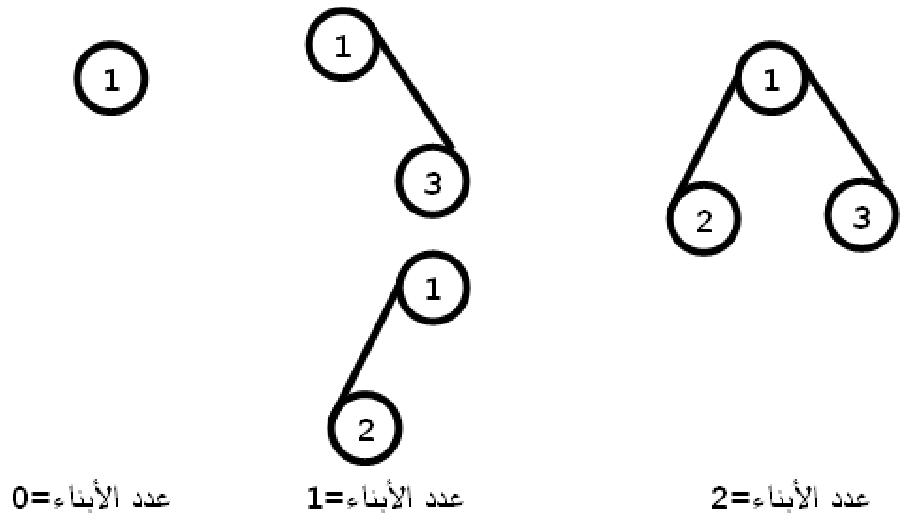


(ب)

العمق 4=(depth)

الأشجار الثنائية Binary Tree

الشجرة الثنائية هي الشجرة التي تحتوى كل عقدة فيها على (**two children**) على الأكثر
يجب أن يكون لكل عقدة في الشجرة الثنائية عدد **0** أو **1** أو **2** من **الأبناء**
والشكل رقم (6) يوضح ذلك



عدد الأبناء بعقدة في الشجرة الثنائية (6) الشكل رقم

كما يمكن أن يكون الابن شجيرة وبالتالي يمكن أن تكون هنالك شجيرتين منفصلتين عن بعضيهما يطلق عليهما الشجيرة اليسرى والشجيرة اليمنى وهما بحد ذاتهما هياكل شجرية ثنائية وبالتالي يكون التعريف متكرراً **Recursive** وهذه ميزة مهمة تسهل عملية البرمجة.
يتميز هيكل الشجرة الثنائية بوجود مؤشري ربط علي الأكثر في أي عقدة أو بمعنى آخر أنه يمكن لعقدة أن تحتوي على ممر واحد أو أثنين أو لا تحتوي على ممر وهي في هذه الحالة تسمى ورقة كما ذكرنا ذلك سابقاً.

التنقل عبر الأشجار الثنائية Binary Tree Traversal

أسلوب التنقل **traversing** عبر الأشجار يقصد به زيارة كل عقدة في الشجرة مرة واحدة وبترتيب معين وذلك بغرض إجراء عملية محددة على البيانات وهو بالتالي غير قاصر على الأشجار الثنائية إذ يمكن تنفيذه على أي هيكل بيانات آخر وعادة ما يتم التنقل عبر الأشجار من خلال ثلاثة أساليب هي:

§ أسلوب التنقل الوسطي **Inorder Traversal** .

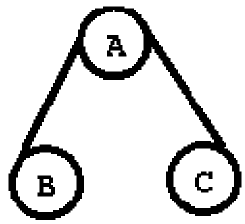
§ أسلوب التنقل البعدي **Postorder Traversal** .

§ أسلوب التنقل القبلي **Preorder Traversal** .

ويعتبر أسلوب التنقل الوسطي **Inorder Traversal** هو الشائع والأكثر استخداماً.

أسلوب التنقل الوسطي Inorder Traversal

يبدأ التنقل في هذا الأسلوب بزيارة الابن الأيسر **L** **ثم** العقدة المراد بيانها **N** ثم الابن الأيمن **R** ويمكن أن نختصر هذا الأسلوب بـ **LNR** وإذا اعتبرنا أن الزيادة تعني طباعة قيمة العقدة وعند تطبيق أسلوب التنقل الوسطي على الشكل التالي



تكون النتيجة هي **BAC**

قد يكون الابن الأيسر أو الأيمن أو كليهما عبارة عن شجيرة فرعية فيتم زيارة الشجيرة الفرعية اليسر ى أولاً وفيها يتم زيارة الابن الأيسر وهكذا ونرى في هذا الأسلوب تكراراً يستوجب استخدام استدعاء النداء الذاتي للدالة **Recursive Function** وستؤدي الدالة ثلاث مهام:

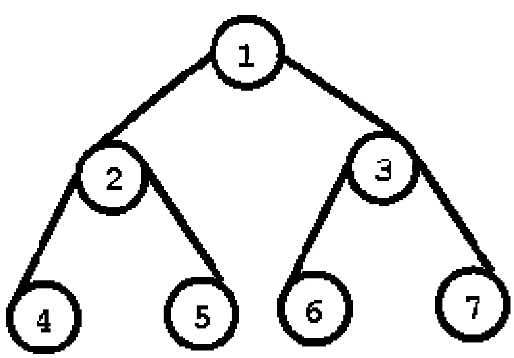
1- التنقل عبر الشجيرة الفرعية اليسر ى للعقدة المدخلة.

2- زيارة العقدة المدخلة.

3- التنقل عبر الشجيرة الفرعية اليمنى للعقدة المدخلة.

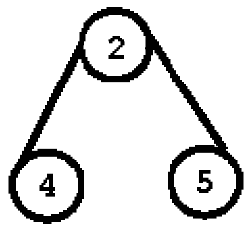
مثال (5)

ما نتيجة الطباعة عند تنفيذ أسلوب التنقل الوسطي على الهياكل الشجرية التالية



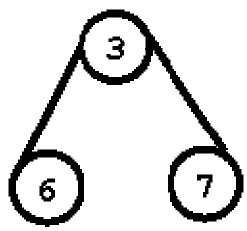
(أ)

نلاحظ أن للجذر **1** ابن أيسر وابن أيمين وهما عبارة عن شجيرة فرعية نبدأ بالشجيرة الفرعية من الجانب الأيسر وفيها نجد أن العقد **2** لها ابن أيسر **4** وابن أيمن **5** وعليه سوف يتم الزيارة الشجيرة في الجانب الأيسر حسب التسلسل التالي



4->2->5

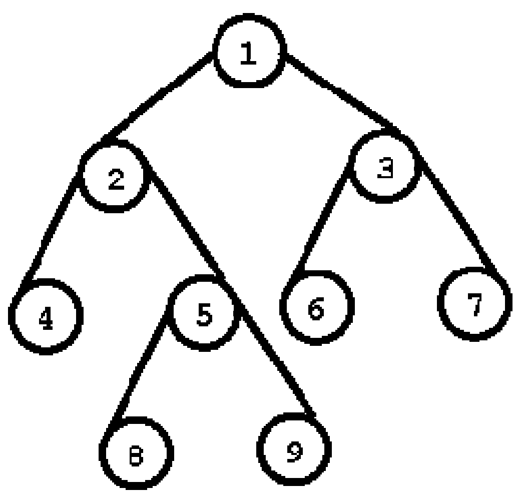
ومن ثم نقوم بزيارة الجذر **1** وبعدها نقوم بزيارة الشجيرة الفرعية من الجانب الأيمن وفيها نجد أن العقد **3** لها ابن أيسر **6** وابن أيمن **7** وعليه سوف يتم الزيارة الشجيرة حسب التسلسل التالي



6->3->7

وعليه تكون النتيجة هي

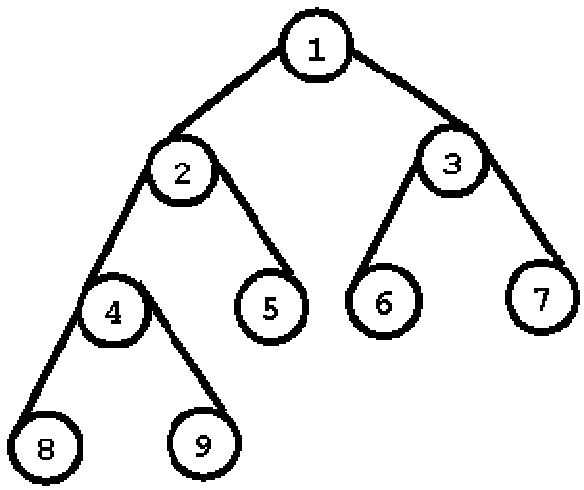
4->2->5->1->6->3->7



(ب)

النتيجة هي

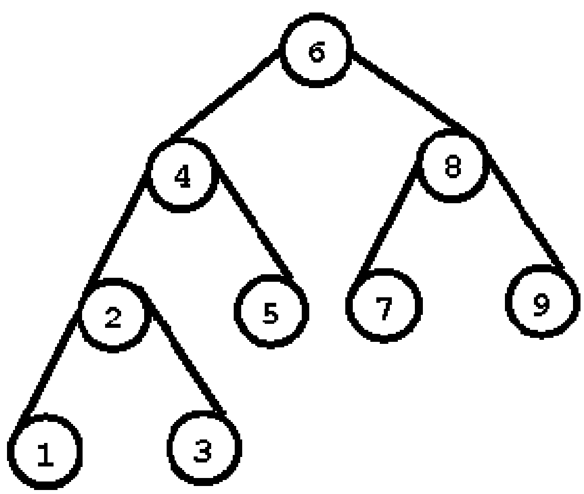
4->2->8->5->9->1->6->3->7



(ج)

النتيجة هي

8->4->9->2->5->1->6->3->7



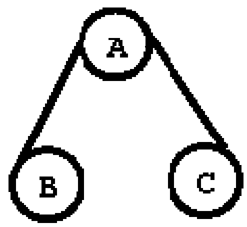
(د)

النتيجة هي

1->2->3->4->5->6->7->8->9

اسلوب التنقل البعدي Postorder Traversal

يبدأ التنقل في هذا الأسلوب بزيارة الابن الأيسر **L ثم** الابن الأيمن **R** من ثم العقدة المراد بيانها **N** ويمكن أن نختصر هذا الأسلوب بـ **LRN** وإذا اعتبرنا أن الزيارة تعني طباعة قيمة العقدة وعند تطبيق أسلوب التنقل البعدي على الشكل التالي



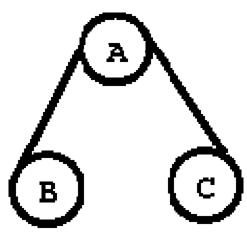
تكون النتيجة هي **BCA**

قد يكون الابن الأيسر أو الأيمن أو كليهما عبارة عن شجيرة فرعية فيتم زيارة الشجيرة الفرعية من الجانب الأيسر أولاً وفيها يتم زيارة الابن الأيسر وهكذا ونرى في هذا الأسلوب تكرار يستوجب استخدام استدعاء النداء الذاتي للدالة **Recursive Function** وستؤدى الدالة ثلاث مهام:

1- التنقل عبر الشجيرة الفرعية من الجانب الأيسر للعقدة المدخلة.
2- التنقل عبر الشجيرة الفرعية من الجانب الأيمن للعقدة المدخلة.
3- زيارة العقدة المدخلة.

أسلوب التنقل القبلي Preorder Traversal

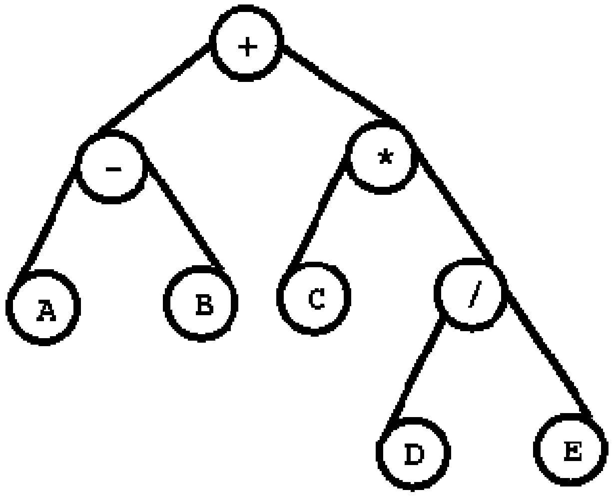
يبدأ التنقل في هذا الأسلوب بزيارة العقدة المراد بيانها **N** ثم الابن الأيسر **L ومن ثم** الابن الأيمن **R** ويمكن أن نختصر هذا الأسلوب بـ **NLR** وإذا اعتبرنا أن الزيارة تعني طباعة قيمة العقدة وعند تطبيق أسلوب التنقل القبلي على الشكل التالي



تكون النتيجة هي **ABC**

مثال (6)

ما نتيجة الطباعة عند تنفيذ أسلوب التنقل الوسطي والبعدي والقبلي علي الاشجار الثنائية التالية

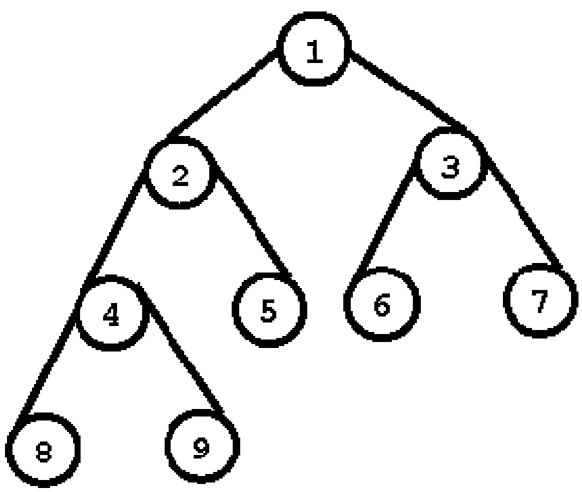


(أ)

نتيجة أسلوب التنقل الوسطي هو **A-B+C*D/E**

نتيجة أسلوب التنقل البعدي هو **AB-CDE/*+**

نتيجة أسلوب التنقل القبلي هو **+-AB*C/DE**



(ب)

نتيجة أسلوب التنقل الوسطي هو

<u>8->4->9->2->5->1->6->3->7</u>

نتيجة أسلوب التنقل البعدي هو

8->9->4->5->2->6->7->3->1

نتيجة أسلوب التنقل القبلي هو

1->2->4->8->9->5->3->6->7

مثال (7)

إذا طبقنا أسلوب التنقل الوسطي على شجرة ثنائية ينتج الترتيب التالي

1->2->3->4->5->6->7->8->9

وإذا طبقنا أسلوب التنقل القبلي على ذات الشجرة ينتج الترتيب التالي

6->4->2->1->3->5->8->7->9

أرسم الشجرة.

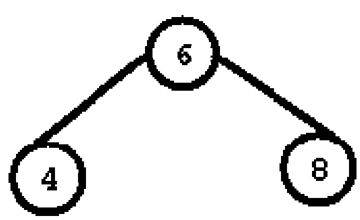
الحل

1- نجد في أسلوب التنقل القبلي أن الجذر هو أول عقدة يتم زيارته وبالتالي يعتبر العقدة 6 هي جذر الشجرة الثنائية.

2- نجد في أسلوب التنقل الوسطي أن الجذر(وهو العقدة 6) يفصل بين الجانب الأيسر للشجرة والجانب الأيمن وعليه فإن العقد 1,2,3,4,5 تمثل الجانب الأيسر للشجرة بينما تمثل العقد 7,8,9 الجانب الأيمن للشجرة.

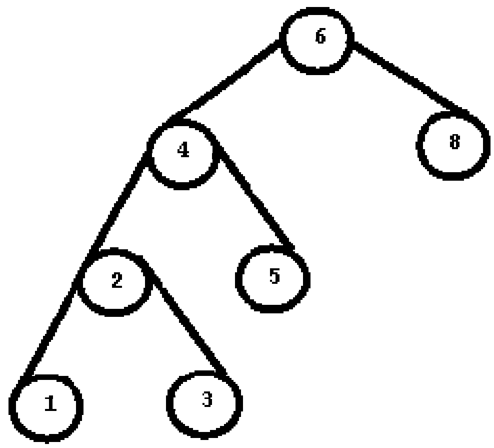
3- تعتبر أول عقدة من العقد التي يتألف منها الجانب الأيسر للشجرة وهي العقد {1,2,3,4,5} في أسلوب التنقل القبلي هي الابن الأيسر للجذر 6 وهي العقدة 4 وأن أول عقدة من العقد التي يتألف منها الجانب الأيمن للشجرة وهي العقد {7,8,9} في أسلوب التنقل القبلي هي الابن الأيمن للجذر 6 وهي العقدة 8 .

وحتى الآن تم تحديد الجذر والابن الأيمن والأيسر كما يوضحه الشكل التالي:



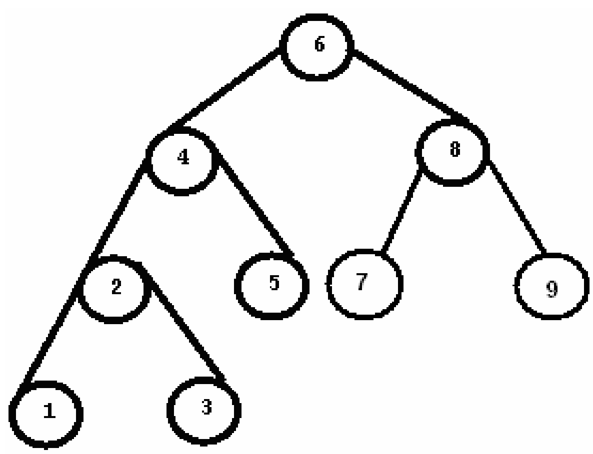
4- يعتبر الابن الأيسر للجذر هو جذر الشجيرة في الجانب الأيسر وعليه فإن العقدة 4 هي جذر الشجيرة في الجانب الأيسر ونجد في أسلوب التنقل الوسطي أن الجذر(وهو العقدة 4) يفصل بين الجانب الأيسر للشجيرة والجانب الأيمن وعليه فإن العقد 1,2,3 تمثل الجانب الأيسر للشجيرة بينما تمثل العقدة 5 الجانب الأيمن للشجيرة الفرعية.

5- تعتبر أول عقدة من العقد التي يتألف منها الجانب الأيسر للشجيرة وهي العقد {1,2,3} في أسلوب التنقل القبلي هي الابن الأيسر للجذر 4 وهي العقدة 2 وهي في نفس الوقت تكون جذراً للعقدتين المتبقيتين وهما {1,3} وحسب أسلوب التنقل الوسطي أن الجذر(وهو العقدة 2) يفصل بين الجانب الأيسر للشجيرة والجانب الأيمن وعليه فإن العقدة 1 تمثل الجانب الأيسر للشجيرة بينما تمثل العقدة 3 الجانب الأيمن



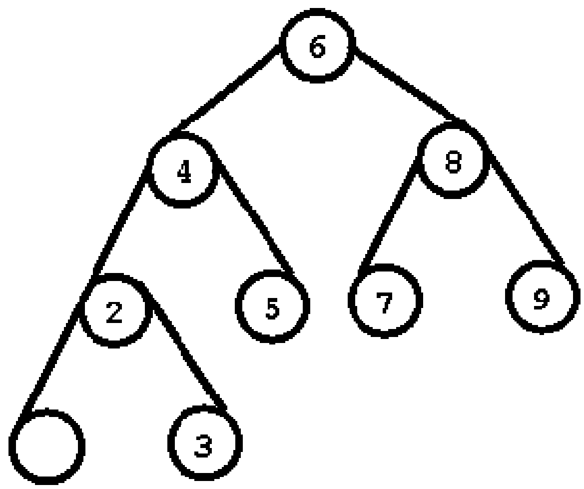
6 - يعتبر الابن الأيمن للجذر هو جذر الشجيرة في الجانب الأيمن وعليه فإن العقدة 8 هي جذر الشجيرة في الجانب الأيمن ونجد في أسلوب التنقل الوسطي أن الجذر (وهو العقدة 8) يفصل بين الجانب الأيسر للشجيرة والجانب الأيمن وعليه فإن العقد 7 تمثل الجانب الأيسر للشجيرة بينما تمثل العقدة 9 الجانب الأيمن للشجيرة الفرعية.

وعليه يصبح الشكل النهائي للشجرة كما يلي :



شجرة البحث الثنائي Binary Search Tree

تعتبر شجرة البحث الثنائي نوعاً معيناً من الأشجار الثنائية تحقق فيه علاقة الابن الأيسر لأي عقدة أقل من أو يساوي العقدة والابن الأيمن أكبر من



بناء الشجرة الثنائية :-

تتكون الشجرة من السجل component الذي يتكون بدوره من ثلاثة حقول أحدهما Data وفيه العنصر المراد وضعه بالشجرة والعنصرين الآخرين Left و right وهما مؤشرين للعقدة اليمين والعقدة الشمال

```
type
  BinarySearchTree = ^Component;
  Component = record
              Data: integer;
              Left, Right: BinarySearchTree
            end;
```

دالة إخلاء الشجرة الثنائية:

تستقبل جذر الشجرة وتضع به القيمة nil بغرض تفريغ الشجرة المعنية

```
function MakeEmptyBinarySearchTree: BinarySearchTree;
begin
  MakeEmptyBinarySearchTree := nil
end;
```

عمل عقدة

ترسل للدالة القيمة المراد وضعها بالدالة ويتم تعريف متغير من نوع الشجرة الثنائية وتوضع القيمة في حقل البيانات الخاص به مع جعل مؤشري اليمين واليسار يشيران إلى عقد فارغة

```
function MakeSingletonBinarySearchTree (Elm: integer):
BinarySearchTree;
var
  Result: BinarySearchTree;
begin
  New (Result);
  Result^.Data := Elm;
  Result^.Left := MakeEmptyBinarySearchTree;
  Result^.Right := MakeEmptyBinarySearchTree;
  MakeSingletonBinarySearchTree := Result
end;
```

التأكد من خلو الشجرة

يتم اختبار جذر الشجرة إذا كان يؤشر إلى nil ترجع الدالة القيمة true وإذا لم تكن تؤشر إلى nil يعني ذلك أن الشجرة غير فارغة

```
function EmptyBinarySearchTree (B: BinarySearchTree): Boolean;
begin
  if B = nil  then EmptyBinarySearchTree := true
else then EmptyBinarySearchTree := false;
end;
```

الإدخال في الشجرة الثنائية

يستقبل هذا الإجراء العنصر المراد إدخاله في الشجرة وجذر الشجرة ا لثنائية ، يتم إرسال العنصر إلى دالة عمل العقدة ثم تتم مقارنة القيمة الجديدة بقيمة جذر الشجرة فإذا كانت أقل من قيمة البيانات بالحذر يتم نداء الدالة نداءاً ذاتياً بالفرع الأيسر وإذا كانت أكبر يتم نداؤها ذاتياً بالفرع الأيمن

```
procedure InsertIntoBinarySearchTree (Elm: integer; var B:
BinarySearchTree);

begin
  if EmptyBinarySearchTree (B) then
    B := MakeSingletonBinarySearchTree (Elm)
  else if Elm < B^.Data then
    InsertIntoBinarySearchTree (Elm, B^.Left)
  else
    InsertIntoBinarySearchTree (Elm, B^.Right)
end;
```

البحث في الشجرة الثنائية

يتم بطريقة مشابهة للإضافة حيث يتم البحث في الجذر فإذا لم نجد جزء بياناته مطابقاً لمفتاح البحث أو العنصر المراد البحث عنه key نختبره مع العنصر فإذا كان أقل منه ننادي دالة البحث نداءاً ذاتياً بالجزء اليسار للعقدة الحالية وإن كان أكبر نناديها بالجزء اليمين. وعندما نجده نرسل العقدة التي تحتويه كإجابة

```
function SearchBinarySearchTree (key: integer; B:
BinarySearchTree;
   var Found: integer): Boolean;
begin
  if EmptyBinarySearchTree (B) then
    SearchBinarySearchTree := False
  else if key < B^.Data then
    SearchBinarySearchTree := SearchBinarySearchTree (key,
B^.Left, Found)
  else if B^.Data < key then
    SearchBinarySearchTree := SearchBinarySearchTree (key,
B^.Right, Found)
  else begin
    SearchBinarySearchTree := True; { because key = B^.Data }
    Found := B^.Data
  end
end;
```

طباعة عناصر القائمة المتصلة

تتم طباعة عناصر القائمة المتصلة بالنداء الذاتي للدالة جهة اليسار وطباعة كل العناصر حتى نصل إلى عقدة فارغة nil ثم ننادي عناصر جهة اليمين بنفس الطريقة

```
procedure PrintBinarySearchTreeData (B: BinarySearchTree);
begin
  if not EmptyBinarySearchTree (B) then begin
    PrintBinarySearchTreeData (B^.Left);
    writeln (B^.Data);
    PrintBinarySearchTreeData (B^.Right)
  end
end;

begin
end.
```

الملفات النصية

الملف النصي Text File هو ملف من الحروف والرموز العادية بدون رسومات أو جداول أو غيرها ويكتب عادة في المفكرة notepad وغالباً ما يكون امتداده txt لذلك يعرف بـ Text File

يتم تعريف الملف النصي كمتغير من نوع text

```
var
  F : text;
```

يختلف الملف عن باقي هياكل البيانات في أنه يتم عمله

الاسم الفيزيائي للملف النصي هو الاسم الحقيقي الذي يحمله الملف على القرص الصلب والاسم المنطقي هو اسم المتغير الذي نستخدمه في البرنامج الرئيسي للإشارة إلى ذلك الملف

لربط الاسم الفيزيائي للملف بالاسم المنطقي نستخدم الشفرة

```
assign(F,'1.txt');
```

وتعني أن المتغير F يشير إلى الملف النصي المسمى (1.txt)
ولا بد من تعيين المسار الصحيح للملف حيث أن ذكر الملف بدون مسار يعني أن الملف سيتم عمله على نفس المجلد الموجود به البرنامج. فمثلاً لعمل الملف على drive D تكون الجملة كالتالي

```
assign(F,'D:\1.txt');
```

وإذا كان الملف بداخل مجلد يسمى programs داخل drive E نكتب

```
assign(F,'E:\programs\1.txt');
```

فتح الملف للكتابة

لإنشاء ملف نصي جديد وفتحه للكتابة عليه نستخدم الشفرة

```
Rewrite(F);
```

وهي تجهز الملف للكتابة عليه كما أنها تمسح كل البيانات الموجودة مسبقاً بالملف

الكتابة على الملف

ولكن قبل كتابة الجملة يجب كتابة اسم الملف writelnتتم بالأمر

```
writeln(f,'This is my first File');
```

إغلاق الملف

من الضروري أن يتم إغلاق الملف بعد إجراء العمليات عليه حتى تنتقل التغييرات التي تمت في RAMأثناء تنفيذ البرنامج إلى القرص الصلب . ويتم ذلك بالعبارة

```
Close(f);
```

مثال توضيحي

ثم نكتب عليهD drive على 1.txtسنجهز ملفاً نسميه

This is my first File
Done with pascal program

على شاشة تنفيذ البرنامج للإشارة إلى أن المهمة file created وبعد ذلك نطبع العبارة ...
قد تمت بنجاح

```
var
  F:text;
begin
    assign(f,'d:\1.txt');
    rewrite(f);
    writeln(f,'This is my first File');
    writeln(f,'Done with pascal program');
    writeln('...file created');
    readln
end.
```

dirve D ثم الدخول إلى My computerلمطالعة نتيجة تنفيذ الملف أعلاه يجب فتح
وستجد أن هنالك ملف نصي يسمى (1) تفتحه بالطريقة العادية وستجد ما كتبت

فتح الملف للقراءة

لطباعة محتويات ملف موجود مسبقاً يجب أولاً ينبغي تجهيزه للقراءة عن طريق العبارة

Reset(f);

متبوعة باسم الملف والمتغير النصي readlnوتتم القراءة من الملف باستخدام العبارة
الذي تتم فيه وضع الجملة المقروءة من الملفstring

readln(f,st);

ولقراءة ما كتبنا في الملف السابق نستخدم البرنامج

```
var
  F:text;
  st:string;
begin
    assign(f,'d:\1.txt');
    reset(f);
    while not eof(f) do
    begin
        readln(f,st);
        writeln(st);
    end;
    close(f);

    readln
end.
```

فتح الملف للإضافة

للإضافة إلى محتويات ملف منشأ مسبقاً يجب أولاً تجهيز الملف للإضافة عن طريق الأمر
Append(f);
ثم بعد ذلك تتم الكتابة عليه بالطريقة العادية علماً بأن الكتابة ستبدأ من نهاية الملف

```
var
  F:text;
begin
    assign(f,'d:\1.txt');
    append(f);
    writeln(f,'this is a pascal program');
    writeln(f,'success of append operation');
    close(f);
    writeln('...file altered');
    readln
end.
```

مثال
اكتب برنامج يطبع الأرقام من 1 إلى 100 في الملف النصي numbers الموجود على drive E

```
ar
  F:text;
  i:integer;
begin
    assign(f,'E:\Numbers.txt');
    rewrite(f);
    for i:= 1 to 100 do
        writeln(f,i);
    close(f);
    writeln('...created');
    readln
end.
```

مثال

اكتب برنامج يقرأ الأعداد من الملف السابق ويوجد مجموعها ووسطها الحسابي ويطبعهما على الشاشة

```
ar
  F:text;
  i,sum,Avg:integer;
begin
    assign(f,'E:\Numbers.txt');
    reset(f);
    sum:=0;
    while not eof(f) do
    begin
        readln(f,i);
        sum:=sum+i
    end;
    close(f);
    Avg:=sum div 100;
    writeln('summation=',sum,' .. and Averrage=',Avg);
    readln
end.
```

مثال

(1) إلى الملف الأول numbers صمم برنامج يضيف محتويات الملف السابق

```
var
  F1,F2:text;
  i,sum,Avg:integer;
begin
    assign(f1,'d:\1.txt');
    assign(f2,'E:\Numbers.txt');
    append(f1);
    reset(f2);
    while not eof(f2) do
    begin
        readln(f2,i);
        writeln(f1,i);
    end;
    close(f1);
    close(f2);
    writeln('Flile content Added');
    readln
end.
```

الملفات النوعية

الملفات النوعية هي ملفات السجلات ، أي ملفات تحتوي على سجلات records ولا يمكن استعراضها إلا عبر برامج باسكال على النقيض من الملفات النصية التي يمكن استعراضها من مستكشف الويندوز.

للتعامل مع الملفات النوعية يجب علينا تجهيز السجلات المراد إدخالها في الملف أولاً ثم وضعها داخل الملف بنفس طريقة وضع النصوص داخل الملف النصي

العمليات على الملف:

يتم ربط الاسم الفيزيائي للملف بالاسم المنطقي له بنفس طريقة الملف النصي

```
assign(F,'d:\Student');
```

ويتم فتحه للكتابة بالأمر rewrite

```
rewrite(F);
```

وفتحه للقراءة بالأمر reset

```
reset(F);
```

وإغلاقه بالأمر close

```
close(F);
```

والإضافة عليه بالأمر append

```
Append(F);
```

ويختلف تعريفه عن تعريف الملف النصي بحيث يجب تعريف السجل أولاً قبل تعريف الملف.
ولنفترض أن لدينا سجل يحتوي على بيانات الطلاب المتكونة من الرقم الجامعي ID والاسم name يجب تعريفه أولاً على الشكل التالي

```
Type
Student=Record
        ID: integer;
        Name: string;
end;
```

ويتم تعريف الملف اعتماداً على السجل بالشكل التالي

```
F:file of Student;
```

وفيما يلي برنامج يقرأ بيانات 5 طلاب ويحفظها في سجلات ثم يخزن السجلات في ملف يسمى student . وبعد ذلك يقرأ بيانات الملف ويطبعها على الشاشة

```
uses crt;
Type
Student=Record
            ID: integer;
            Name: string;
end;
var
  St:student;
  F:file of Student;
  i: integer;
begin
  clrscr;
  assign(F,'d:\Student');
  rewrite(F);
  for i:=1 to 5 do
  begin
      writeln('Enter student No');
      read(st.ID);
      writeln('Enter Student Name');
      read(st.name);
      write(F,st);
  end;
  close(F);
  reset(F);
  while not eof(F) do
  begin
      read(F,st);
      writeln(st.ID,'  ',st.name);
  end;
  close(F);
  readln;
end.
```